

MICROCHIP

Embedded Control Handbook
Volume 1

SERVING A COMPLEX AND COMPETITIVE WORLD WITH FIELD-PROGRAMMABLE EMBEDDED CONTROL SYSTEM SOLUTIONS

© 1997 Microchip Technology Inc.

April 1997 DS00092D

"All rights reserved. Copyright © 1997, Microchip Technology Incorporated, USA. Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights."

The Microchip name, logo, KEELOQ, PIC, PICMASTER, PICSTART, PICSEE, PRO MATE, TrueGauge and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Inc. in the U.S.A. and other countries.

PICmicro, the TrueGauge logo, *fuzzy*LAB, Smart Serial, Total Endurance, and The Emerging World Standard are trademarks of Microchip Technology Inc.

SQTP is a service mark of Microchip Technology Inc.

All other trademarks mentioned herein are property of their respective companies.

© 1997 Microchip Technology Inc.

MICROCHIP

MICROCHIP TECHNOLOGY INC. COMPANY PROFILE 1

PIC16C5X APPLICATION NOTES 2

PIC16CXXX APPLICATION NOTES 3

PIC17CXXX APPLICATION NOTES 4

PICmicro™ MICROCONTROLLER APPLICATION NOTES 5

SECURE DATA PRODUCTS APPLICATION NOTES 6

SERIAL EEPROM APPLICATION NOTES 7

QUALITY AND RELIABLILITY 8

TECHNICAL BRIEFS 9

REFERENCE DESIGNS 10

DEVELOPMENT TOOLS 11

OFFICE LOCATIONS 12

© 1997 Microchip Technology Inc.

TO OUR VALUED CUSTOMERS:

Welcome to the Microchip Technology Inc. ***Embedded Control Handbook (ECHB) - Volume 1.*** The *ECHB - Volume 1* is the first 'Volume' in our library system of PICmicro™ 8-Bit One-Time-Programmable (OTP) Microcontroller, Nonvolatile Memory, Secure Data Products, and other product application notes, technical briefs, and reference designs. Volume 1 replaces the 1994/1995 ECHB (released in September 1994) and 1995/1996 ECHB Update I (released in September 1995).

Microchip will continue publishing application notes, technical briefs and reference designs in a series of supplemental handbooks called 'Updates'. Updates will be published annually, providing an uninterrupted flow of current application notes, technical briefs and reference designs for our customers' convenience and use. These Updates, with revised and new documents, will be incorporated into future Volumes as appropriate.

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip product. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new Volumes and Updates are introduced. We welcome your feedback.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department at facsimile 602.917.4150.

© 1997 Microchip Technology Inc.

MICROCHIP

Table of Contents

PAGE

SECTION 1 COMPANY PROFILE

Introduction the The Embedded Control Solutions Company 1-1

SECTION 2 PIC16C5X APPLICATION NOTES

Implementation of an Asynchronous Serial I/O - AN510 2-1
PLD Replacement - AN511 2-19
Implementing Ohmmeter/Temperature Sensor - AN512 2-41
Analog to Digital Conversion Using a PIC16C54 - AN513 2-47
Software Interrupt Techniques - AN514 2-53
Implementing a Simple Serial Mouse Controller - AN519 2-57
A Comparison of 8-Bit Microcontrollers - AN520 2-69
Interfacing to AC Power Lines - AN521 2-79
Power-up Considerations - AN522 2-81
Software Stack Management - AN527 2-85
Implementing Wake-up on Key Stroke - AN528 2-89
Multiplexing LED Drive and a 4x4 Keypad Sampling - AN529 2-95
Intelligent Remote Positioner (Motor Control) - AN531 2-121
Using a PIC16C5X as a Smart I^2C^{TM} Peripheral - AN541 2-135
Using PIC16C5X Microcontrollers as LCD Drivers - AN563 2-151
PIC16C54A EMI Results - AN577 2-165
Implementing Long Calls - AN581 2-171
Macros for Page and Bank Switching - AN586 2-175
A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs - AN590 2-197
Frequency Counter Using PIC16C5X - AN592 2-209
Serial Port Routines Without Using Timer0 - AN593 2-219
Modifying PIC16C54A Code for the PIC16C58A - AN618 2-225
Code Development for the PIC16C52 - AN641 2-233
D/A Conversion Using PWM and R-2R Ladders to Generate Sine and DTMF Waveforms - AN655 2-235
Decoding Infrared Remote Controls Using a PIC16C5X Microcontroller - AN657 2-253

© 1997 Microchip Technology Inc.

Table of Contents (Continued)

PAGE

SECTION 3 PIC16CXXX APPLICATION NOTES

Using the Analog-to-Digital (A/D) Converter - AN546 3-1
Implementing Wake-up on Key Stroke - AN552 3-21
Software Implementation of I²C™ Bus Master - AN554 3-25
Software Implementation of Asynchronous Serial I/O - AN555 3-85
Four Channel Digital Voltmeter with Display and Keyboard - AN557 3-121
Using the PORTB Interrupt on Change as an External Interrupt - AN566 3-149
Use of the SSP Module in the I²C™ Multi-Master Environment - AN578 3-153
Using the 8-Bit Parallel Slave Port - AN579 3-169
Using Timer1 in Asynchronous Clock Mode - AN580 3-177
Low-Power Real-Time Clock - AN582 3-181
Interfacing PICmicro™ Microcontrollers to an LCD Module - AN587 3-205
A PC-Based Development Programmer for the PIC16C84 - AN589 3-237
Apple® Desktop Bus (ADB™) - AN591 3-243
Using the CCP Module(s) - AN594 3-277
Implementing Ultrasonic Ranging - AN597 3-301
Air Flow Control Using Fuzzy Logic - AN600 3-319
Resistance and Capacitance Meter Using a PIC16C622 - AN611 3-339
Digital Signal Processing with the PIC16C74 - AN616 3-373
PIC14C000 Calibration Parameters - AN621 3-405
PIC14C000 A/D Theory and Implementation - AN624 3-411
Lead-Acid Battery Charger Implementation Using PIC14C000 - AN626 3-421
Adaptive Differential Pulse Code Modulation using PICmicro™ Microcontrollers - AN643 3-451
Yet Another Clock Featuring the PIC16C924 - AN649 3-491
PWM, a Software Solution for the PIC16CXXX - AN654 3-523
In-Circuit Serial Programming of Calibration Parameters Using a PICmicro™ Microcontroller- AN656 3-535
LCD Fundamentals Using PIC16C92X Microcontrollers - AN658 3-557
Smart Battery Charger with SMBus Interface - AN667 3-579

SECTION 4 PIC17CXXX APPLICATION NOTES

Servo Control of a DC-Brush Motor - AN532 4-1
Saving and Restoring Status on Interrupt (Implementing a Parameter Stack) - AN534 4-141
Using PWM to Generate Analog Output - AN538 4-143
Frequency and Resolution Options for PWM Outputs - AN539 4-145
Implementing IIR Digital Filters - AN540 4-157
Implementation of Fast Fourier Transforms - AN542 4-177
Tone Generation - AN543 4-199
Math Utility Routines - AN544 4-209
Using the Capture Module - AN545 4-259
Serial Port Utilities - AN547 4-283
Implementing Table Read and Table Write - AN548 4-295
Using the PWM - AN564 4-299
Implementation of the Data Encryption Standard Using PIC17C42 - AN583 4-311

© 1997 Microchip Technology Inc.

MICROCHIP

Table of Contents (Continued)

PAGE

SECTION 5 PIC16/17 APPLICATION NOTES

PIC16C5X / PIC16CXXX Math Utility Routines - AN526 5-1
Implementing a Table Read - AN556 5-95
Techniques to Disable Global Interrupts - AN576 5-99
A Real-Time Operating System for PICmicro™ Microcontrollers - AN585 5-105
PICmicro™ Microcontroller Oscillator Design Guide - AN588 5-143
Low Power Design Using PICmicro™ Microcontroller - AN606 5-161
Power-up Trouble Shooting - AN607 5-177
Clock Design Using Low Power/Cost Techniques - AN615 5-199

SECTION 6 SECURE DATA PRODUCTS APPLICATION NOTES

Converting NTQ104/105/106 Designs to HCS200/300s - AN644 6-1
PIC16C57 Based Code Hopping Security System - AN645 6-7
KEELOQ® Code Hopping Decoder Using a PIC16C56 - AN661 6-19
KEELOQ® Code Hopping Decoder Using Secure Learn - AN662 6-33
KEELOQ® Simple Code Hopping Decoder - AN663 6-49
Using KEELOQ® to Generate Hopping Passwords - AN665 6-63

SECTION 7 SERIAL EEPROM APPLICATION NOTES

Communicating with the I²C™ Bus Using the PIC16C5X - AN515 7-1
24C01A Compatibility Issue and Its Mobility for Memory Upgrade - AN517 7-11
Interfacing 93CX6 Serial EEPROMs to PIC16C5X Microcontrollers - AN530 7-13
Logic Powered Serial EEPROMs - AN535 7-35
Basic Serial EEPROM Operation - AN536 7-45
Serial EEPROM Endurance - AN537 7-59
1.8 Volt Technology – Benefits - AN550 7-67
Serial EEPROM Solutions vs. Parallel Solutions - AN551 7-69
Using the 24xx65 and 25xx32 with Stand-alone PIC16C54 Code - AN558 7-73
Optimizing Serial Bus Operations with Proper Write Cycle Times - AN559 7-117
Using the 93LC56 and 93LC66 - AN560 7-121
Using the Microchip Endurance Predictive Software - AN562 7-141
Interfacing the 24LCXXB Serial EEPROMs to the PIC16C54 - AN567 7-145
Questions and Answers Concerning Serial EEPROMs - AN572 7-173
EEPROM Endurance Tutorial - AN601 7-177
How to get 10 Million Cycles Out of Your Microchip Serial EEPROM - AN602 7-181
Converting to 24LCXXB and 93LCXX Serial EEPROMs - AN608 7-183
Interfacing Microchip Serial EEPROMs to Motorola® 68HC11 Microcontroller - AN609 7-185
Using the 24LC21 Dual Mode Serial EEPROM - AN610 7-193
Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI™ Ports - AN613 7-199
Interfacing the 8051 with 2-wire Serial EEPROMs - AN614 7-213
Interfacing the 93XX76 and 93XX86 to a PIC16C5X - AN619 7-223
Interfacing Motorola 68HC11 to Microchip SPI™ Serial EEPROMS - AN646 7-225
Interfacing Microchip PIC16C64/74 to Microchip SPI™ Serial EEPROMs - AN647 7-231
Interfacing Microchip PIC16C54 to Microchip SPI™ Serial EEPROMs - AN648 7-235

SECTION 8 QUALITY AND RELIABILITY

Improving the Susceptibility of an Application to ESD - AN595 8-1
Plastic Packaging and the Effects of Surface Mount Soldering Techniques - AN598 8-19
Continuous Improvement - AN603 8-27

© 1997 Microchip Technology Inc.

Table of Contents (Continued)

PAGE

SECTION 9 TECHNICAL BRIEFS

Secure Learning RKE Systems Using KEELOQ® Encoders - TB001 9-1
An Introduction to KEELOQ® Code Hopping - TB003 9-7
Automatic Calibration of the WDT Time-out Period - TB004 9-17
Using External RAM with PIC17CXX Devices - TB005 9-23
Transformerless Power Supply - TB008 9-31
Adding a Simple 4-channel 8-bit A/D to a PIC17C4X - TB010 9-33
Using SRAM With A PIC16CXX - TB011 9-45

SECTION 10 REFERENCE DESIGNS

Uninterruptible Power Supply Reference Design Based on PIC17C43 - RD001 10-1
Intelligent Battery Charger Reference Design Based on PIC16C7X - RD002 10-3
Watt-Hour Meter Reference Design Based on PIC16C924 - RD003 10-5

SECTION 11 DEVELOPMENT TOOLS

Integrated Development Environment Product Brief 11-9
Universal PICmicro™ Microcontroller Assembler Software 11-11
Software Simulator Product Brief 11-13
Universal C Compiler for PICmicro™ MCUs Product Brief 11-15
Application Code Generator Product Brief 11-17
Fuzzy Logic Development System for PICmicro™ 11-19
CE Compliant In-Circuit Emulator Product Brief 11-21
Full-Featured Modular In-Circuit Emulator 11-23
Low-Cost PIC16CXXX In-Circuit Emulator Product Brief 11-25
Universal Device Programmer Product Brief 11-27
Low Cost Development Kit Product Brief 11-29
Ultra Low Cost Development Kit Product Brief 11-31
KEELOQ HCSXXX Device Programmers 11-33
KEELOQ HCSXXX Evaluation Kit Product Brief 11-35
HCSXXX Code Hopping Technology 11-35
Low-Cost PICmicro™ Demonstration Board 11-37
Low-Cost PIC16CXX Demonstration Board 11-39
Low-Cost PIC16C9XX Demonstration Board 11-41
Microchip Serial EEPROM Designer's Kit 11-43
Microchip Serial EEPROM Endurance Model 11-45

SECTION 12 OFFICE LOCATIONS

Factory Representatives 12-1
Distributors 12-7
Factory Sales 12-21

© 1997 Microchip Technology Inc.

MICROCHIP

Cross Reference Guide to Application Notes - Alphabetical

		PAGE
1.8 Volt Technology – Benefits	AN550	7-67
24C01A Compatibility Issue and Its Mobility for Memory Upgrade	AN517	7-11
A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs	AN590	2-197
A Comparison of 8-Bit Microcontrollers	AN520	2-69
A Real-Time Operating System for PICmicro™ Microcontrollers	AN585	5-105
A PC-Based Development Programmer for the PIC16C84	AN589	3-237
Adaptive Differential Pulse Code Modulation using PICmicro™ Microcontrollers	AN643	3-451
Adding a Simple 4-channel 8-bit A/D to a PIC17C4X	TB010	9-33
Air Flow Control Using Fuzzy Logic	AN600	3-319
An Introduction to KEELOQ® Code Hopping	TB003	9-7
Analog to Digital Conversion Using a PIC16C54	AN513	2-47
Apple® Desktop Bus (ADB™)	AN591	3-243
Automatic Calibration of the WDT Time-out Period	TB004	9-17
Basic Serial EEPROM Operation	AN536	7-45
Clock Design Using Low Power/Cost Techniques	AN615	5-199
Code Development for the PIC16C52	AN641	2-233
Communicating with the I^2C™ Bus Using the PIC16C5X	AN515	7-1
Continuous Improvement	AN603	8-27
Converting NTQ104/105/106 Designs to HCS200/300s	AN644	6-1
Converting to 24LCXXB and 93LCXX Serial EEPROMs	AN608	7-183
D/A Conversion Using PWM and R-2R Ladders to Generate Sine and DTMF Waveforms	AN655	2-235
Decoding Infrared Remote Controls Using a PIC16C5X Microcontroller	AN657	2-253
Digital Signal Processing with the PIC16C74	AN616	3-373
EEPROM Endurance Tutorial	AN601	7-177
Four Channel Digital Voltmeter with Display and Keyboard	AN557	3-121
Frequency and Resolution Options for PWM Outputs	AN539	4-145
Frequency Counter Using PIC16C5X	AN592	2-209
How to get 10 Million Cycles Out of Your Microchip Serial EEPROM	AN602	7-181
Implementation of an Asynchronous Serial I/O	AN510	2-1
Implementation of Fast Fourier Transforms	AN542	4-177
Implementation of the Data Encryption Standard Using PIC17C42	AN583	4-311
Implementing a Simple Serial Mouse Controller	AN519	2-57
Implementing a Table Read	AN556	5-95
Implementing IIR Digital Filters	AN540	4-157
Implementing Long Calls	AN581	2-171
Implementing Ohmmeter/Temperature Sensor	AN512	2-41
Implementing Table Read and Table Write	AN548	4-295
Implementing Ultrasonic Ranging	AN597	3-301
Implementing Wake-up on Key Stroke	AN528	2-89
Implementing Wake-up on Key Stroke	AN552	3-21
Improving the Susceptibility of an Application to ESD	AN595	8-1
In-Circuit Serial Programming of Calibration Parameters Using a PIC16CXXX	AN656	3-535
Intelligent Battery Charger Reference Design Based on PIC16C7X	RD002	10-3
Intelligent Remote Positioner (Motor Control)	AN531	2-121
Interfacing 93CX6 Serial EEPROMs to PIC16C5X Microcontrollers	AN530	7-13
Interfacing Microchip PIC16C54 to Microchip SPI™ Serial EEPROMs	AN648	7-235
Interfacing Microchip PIC16C64/74 to Microchip SPI™ Serial EEPROMs	AN647	7-231
Interfacing Microchip Serial EEPROMs to Motorola® 68HC11 Microcontroller	AN609	7-185
Interfacing Motorola 68HC11 to Microchip SPI™ Serial EEPROMs	AN646	7-225
Interfacing the 24LCXXB Serial EEPROMs to the PIC16C54	AN567	7-145
Interfacing the 8051 with 2-wire Serial EEPROMs	AN614	7-213
Interfacing the 93XX76 and 93XX86 to a PIC16C5X	AN619	7-223
Interfacing to AC Power Lines	AN521	2-79

© 1997 Microchip Technology Inc.

Cross Reference Guide to Application Notes - Alphabetical (Continued)

		PAGE
Interfacing PICmicro™ Microcontrollers to an LCD Module	AN587	3-205
KEELOQ® Code Hopping Decoder Using a PIC16C56	AN661	6-19
KEELOQ® Code Hopping Decoder Using Secure Learn	AN662	6-33
KEELOQ® Simple Code Hopping Decoder	AN663	6-49
LCD Fundamentals Using PIC16C92X Microcontrollers	AN658	3-557
Lead-Acid Battery Charger Implementation Using PIC14C000	AN626	3-421
Logic Powered Serial EEPROMs	AN535	7-35
Low Power Design Using PICmicro™ Microcontrollers	AN606	5-161
Low-Power Real-Time Clock	AN582	3-181
Macros for Page and Bank Switching	AN586	2-175
Math Utility Routines	AN544	4-209
Modifying PIC16C54A Code for the PIC16C58A	AN618	2-225
Multiplexing LED Drive and a 4x4 Keypad Sampling	AN529	2-95
Optimizing Serial Bus Operations with Proper Write Cycle Times	AN559	7-117
PIC14C000 A/D Theory and Implementation	AN624	3-411
PIC14C000 Calibration Parameters	AN621	3-405
PICmicro™ Microcontroller™ Oscillator Design Guide	AN588	5-143
PIC16C54A EMI Results	AN577	2-165
PIC16C57 Based Code Hopping Security System	AN645	6-7
PIC16C5X / PIC16CXXX Math Utility Routines	AN526	5-1
Plastic Packaging and the Effects of Surface Mount Soldering Techniques	AN598	8-19
PLD Replacement	AN511	2-19
Power-up Considerations	AN522	2-81
Power-up Trouble Shooting	AN607	5-177
PWM, a Software Solution for the PIC16CXXX	AN654	3-523
Questions and Answers Concerning Serial EEPROMs	AN572	7-173
Resistance and Capacitance Meter Using a PIC16C622	AN611	3-339
Saving and Restoring Status on Interrupt (Implementing a Parameter Stack)	AN534	4-141
Secure Learning RKE Systems Using KEELOQ® Encoders	TB001	9-1
Serial EEPROM Endurance	AN537	7-59
Serial EEPROM Solutions vs. Parallel Solutions	AN551	7-69
Serial Port Routines Without Using Timer0	AN593	2-219
Serial Port Utilities	AN547	4-283
Servo Control of a DC-Brush Motor	AN532	4-1
Sine and DTMF Waveforms	AN655	2-235
Smart Battery Charger with SMBus Interface	AN667	3-579
Software Implementation of Asynchronous Serial I/O	AN555	3-85
Software Implementation of I²C™ Bus Master	AN554	3-25
Software Interrupt Techniques	AN514	2-53
Software Stack Management	AN527	2-85
Techniques to Disable Global Interrupts	AN576	5-99
Tone Generation	AN543	4-199
Transformerless Power Supply	TB008	9-31
Uninterruptible Power Supply Reference Design Based on PIC17C43	RD001	10-1
Use of the SSP Module in the I²C™ Multi-Master Environment	AN578	3-153
Using a PIC16C5X as a Smart I²C™ Peripheral	AN541	2-135
Using External RAM with PIC17CXX Devices	TB005	9-23
Using KEELOQ to Generate Hopping Passwords	AN665	6-63
Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI™ Ports	AN613	7-199
Using PIC16C5X Microcontrollers as LCD Drivers	AN563	2-151
Using PWM to Generate Analog Output	AN538	4-143
Using SRAM With A PIC16CXX	TB011	9-45
Using the 24LC21 Dual Mode Serial EEPROM	AN610	7-193

© 1997 Microchip Technology Inc.

Cross Reference Guide to Application Notes - Alphabetical (Continued)

		PAGE
Using the 24xx65 and 25xx32 with Stand-alone PIC16C54 Code	AN558	7-73
Using the 8-Bit Parallel Slave Port	AN579	3-169
Using the 93LC56 and 93LC66	AN560	7-121
Using the Analog-to-Digital (A/D) Converter	AN546	3-1
Using the Capture Module	AN545	4-259
Using the CCP Module(s)	AN594	3-277
Using the Microchip Endurance Predictive Software	AN562	7-141
Using the PORTB Interrupt on Change as an External Interrupt	AN566	3-149
Using the PWM	AN564	4-299
Using Timer1 in Asynchronous Clock Mode	AN580	3-177
Watt-Hour Meter Reference Design Based on PIC16C924	RD003	10-5
Yet Another Clock Featuring the PIC16C924	AN649	3-491

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

MICROCHIP

Cross Reference Guide to Application Notes - Numerical

PAGE

AN510 ..... Implementation of an Asynchronous Serial I/O 2-1
AN511 ..... PLD Replacement 2-19
AN512 ..... Implementing Ohmmeter/Temperature Sensor 2-41
AN513 ..... Analog to Digital Conversion Using a PIC16C54 2-47
AN514 ..... Software Interrupt Techniques 2-53
AN515 ..... Communicating with the I²C™ Bus Using the PIC16C5X 7-1
AN517 ..... 24C01A Compatibility Issue and Its Mobility for Memory Upgrade 7-11
AN519 ..... Implementing a Simple Serial Mouse Controller 2-57
AN520 ..... A Comparison of 8-Bit Microcontrollers 2-69
AN521 ..... Interfacing to AC Power Lines 2-79
AN522 ..... Power-up Considerations 2-81
AN526 ..... PIC16C5X / PIC16CXXX Math Utility Routines 5-1
AN527 ..... Software Stack Management 2-85
AN528 ..... Implementing Wake-up on Key Stroke 2-89
AN529 ..... Multiplexing LED Drive and a 4x4 Keypad Sampling 2-95
AN530 ..... Interfacing 93CX6 Serial EEPROMs to PIC16C5X Microcontrollers 7-13
AN531 ..... Intelligent Remote Positioner (Motor Control) 2-121
AN532 ..... Servo Control of a DC-Brush Motor 4-1
AN534 ..... Saving and Restoring Status on Interrupt (Implementing a Parameter Stack) 4-141
AN535 ..... Logic Powered Serial EEPROMs 7-35
AN536 ..... Basic Serial EEPROM Operation 7-45
AN537 ..... Serial EEPROM Endurance 7-59
AN538 ..... Using PWM to Generate Analog Output 4-143
AN539 ..... Frequency and Resolution Options for PWM Outputs 4-145
AN540 ..... Implementing IIR Digital Filters 4-157
AN541 ..... Using a PIC16C5X as a Smart I²C™ Peripheral 2-135
AN542 ..... Implementation of Fast Fourier Transforms 4-177
AN543 ..... Tone Generation 4-199
AN544 ..... Math Utility Routines 4-209
AN545 ..... Using the Capture Module 4-259
AN546 ..... Using the Analog-to-Digital (A/D) Converter 3-1
AN547 ..... Serial Port Utilities 4-283
AN548 ..... Implementing Table Read and Table Write 4-295
AN550 ..... 1.8 Volt Technology – Benefits 7-67
AN551 ..... Serial EEPROM Solutions vs. Parallel Solutions 7-69
AN552 ..... Implementing Wake-up on Key Stroke 3-21
AN554 ..... Software Implementation of I²C™ Bus Master 3-25
AN555 ..... Software Implementation of Asynchronous Serial I/O 3-85
AN556 ..... Implementing a Table Read 5-95
AN557 ..... Four Channel Digital Voltmeter with Display and Keyboard 3-121
AN558 ..... Using the 24xx65 and 25xx32 with Stand-alone PIC16C54 Code 7-73
AN559 ..... Optimizing Serial Bus Operations with Proper Write Cycle Times 7-117
AN560 ..... Using the 93LC56 and 93LC66 7-121
AN562 ..... Using the Microchip Endurance Predictive Software 7-141
AN563 ..... Using PIC16C5X Microcontrollers as LCD Drivers 2-151
AN564 ..... Using the PWM 4-299
AN566 ..... Using the PORTB Interrupt on Change as an External Interrupt 3-149
AN567 ..... Interfacing the 24LCXXB Serial EEPROMs to the PIC16C54 7-145
AN572 ..... Questions and Answers Concerning Serial EEPROMs 7-173
AN576 ..... Techniques to Disable Global Interrupts 5-99
AN577 ..... PIC16C54A EMI Results 2-165
AN578 ..... Use of the SSP Module in the I²C™ Multi-Master Environment 3-153
AN579 ..... Using the 8-Bit Parallel Slave Port 3-169

© 1997 Microchip Technology Inc.

Cross Reference Guide to Application Notes - Numerical (Continued)

PAGE

AN580 ..... Using Timer1 in Asynchronous Clock Mode 3-177
AN581 ..... Implementing Long Calls 2-171
AN582 ..... Low-Power Real-Time Clock 3-181
AN583 ..... Implementation of the Data Encryption Standard Using PIC17C42 4-311
AN585 ..... A Real-Time Operating System for PICmicro™ Microcontrollers 5-105
AN586 ..... Macros for Page and Bank Switching 2-175
AN587 ..... Interfacing PICmicro™ Microcontrollers to an LCD Module 3-205
AN588 ..... PICmicro™ Microcontroller Oscillator Design Guide 5-143
AN589 ..... A PC-Based Development Programmer for the PIC16C84 3-237
AN590 ..... A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs 2-197
AN591 ..... Apple® Desktop Bus (ADB™) 3-243
AN592 ..... Frequency Counter Using PIC16C5X 2-209
AN593 ..... Serial Port Routines Without Using Timer0 2-219
AN594 ..... Using the CCP Module(s) 3-277
AN595 ..... Improving the Susceptibility of an Application to ESD 8-1
AN597 ..... Implementing Ultrasonic Ranging 3-301
AN598 ..... Plastic Packaging and the Effects of Surface Mount Soldering Techniques 8-19
AN600 ..... Air Flow Control Using Fuzzy Logic 3-319
AN601 ..... EEPROM Endurance Tutorial 7-177
AN602 ..... How to get 10 Million Cycles Out of Your Microchip Serial EEPROM 7-181
AN603 ..... Continuous Improvement 8-27
AN606 ..... Low Power Design Using PICmicro™ Microcontrollers 5-161
AN607 ..... Power-up Trouble Shooting 5-177
AN608 ..... Converting to 24LCXXB and 93LCXX Serial EEPROMs 7-183
AN609 ..... Interfacing Microchip Serial EEPROMs to Motorola® 68HC11 Microcontroller 7-185
AN610 ..... Using the 24LC21 Dual Mode Serial EEPROM 7-193
AN611 ..... Resistance and Capacitance Meter Using a PIC16C622 3-339
AN613 ..... Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI™ Ports 7-199
AN614 ..... Interfacing the 8051 with 2-wire Serial EEPROMs 7-213
AN615 ..... Clock Design Using Low Power/Cost Techniques 5-199
AN616 ..... Digital Signal Processing with the PIC16C74 3-373
AN618 ..... Modifying PIC16C54A Code for the PIC16C58A 2-225
AN619 ..... Interfacing the 93XX76 and 93XX86 to a PIC16C5X 7-223
AN621 ..... PIC14C000 Calibration Parameters 3-405
AN624 ..... PIC14C000 A/D Theory and Implementation 3-411
AN626 ..... Lead-Acid Battery Charger Implementation Using PIC14C000 3-421
AN641 ..... Code Development for the PIC16C52 2-233
AN643 ..... Adaptive Differential Pulse Code Modulation using PICmicro™ Microcontrollers 3-451
AN644 ..... Converting NTQ104/105/106 Designs to HCS200/300s 6-1
AN645 ..... PIC16C57 Based Code Hopping Security System 6-7
AN646 ..... Interfacing Motorola 68HC11 to Microchip SPI™ Serial EEPROMs 7-225
AN647 ..... Interfacing Microchip PIC16C64/74 to Microchip SPI™ Serial EEPROMs 7-231
AN648 ..... Interfacing Microchip PIC16C54 to Microchip SPI™ Serial EEPROMs 7-235
AN649 ..... Yet Another Clock Featuring the PIC16C924 3-491
AN654 ..... PWM, a Software Solution for the PIC16CXXX 3-523
AN655 ..... D/A Conversion Using PWM and R-2R Ladders to Generate Sine and DTMF Waveforms 2-235
AN656 ..... In-Circuit Serial Programming of Calibration Parameters Using a PIC16CXXX 3-535
AN657 ..... Decoding Infrared Remote Controls Using a PIC16C5X Microcontroller 2-253
AN658 ..... LCD Fundamentals Using PIC16C92X Microcontrollers 3-557
AN661 ..... Code Hopping Decoder Using a PIC16C56 6-19
AN662 ..... Code Hopping Decoder Using Secure Learn 6-33
AN663 ..... Simple Code Hopping Decoder 6-49
AN665 ..... Using KEELOQ to Generate Hopping Passwords 6-63

© 1997 Microchip Technology Inc.

Cross Reference Guide to Application Notes - Numerical (Continued)

		PAGE
AN667	Smart Battery Charger with SMBus Interface	3-579
RD001	Uninterruptible Power Supply Reference Design Based on PIC17C43	10-1
RD002	Intelligent Battery Charger Reference Design Based on PIC16C7X	10-3
RD003	Watt-Hour Meter Reference Design Based on PIC16C924	10-5
TB001	Secure Learning RKE Systems Using KEELOQ® Encoders	9-1
TB003	An Introduction to KEELOQ® Code Hopping	9-7
TB004	Automatic Calibration of the WDT Time-out Period	9-17
TB005	Using External RAM with PIC17CXX Devices	9-23
TB008	Transformerless Power Supply	9-31
TB010	Adding a Simple 4-channel 8-bit A/D to a PIC17C4X	9-33
TB011	Using SRAM With A PIC16CXX	9-45

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

SECTION 1
COMPANY PROFILE

Introduction to The Embedded Control Solutions Company® 1-1

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

Microchip Technology Inc.

Introduction to The Embedded Control Solutions Company®

Since its inception, Microchip Technology has focused its resources on delivering innovative semiconductor products to the global embedded control marketplace. To do this, we have focused our technology, engineering, manufacturing and marketing resources on two synergistic product lines: field-programmable 8-bit PICmicro™ microcontrollers (MCUs) and high-endurance Serial EEPROMs. Today, Microchip's expanding product portfolio is aimed at delivering a more comprehensive array of high-value solutions to a growing base of customers.

Highlights

Inside Microchip Technology you will find:

- An experienced executive team focused on innovation and committed to listening to our customers
- A focus on providing high-performance, cost-effective, field-programmable embedded control solutions
- 8-bit RISC field-programmable OTP and ROM MCUs
- The world's first 8-pin MCU
- Patented KEELOQ® code hopping technology products
- QuickASIC™ gate array conversion technology
- A full family of leadership Serial EEPROM products plus Parallel EEPROMs and EPROMs
- A variety of end-user Application-Specific Standard Products (ASSP)
- Fully integrated manufacturing capabilities
- A global network of manufacturing and customer support facilities
- A unique corporate culture dedicated to continuous improvement
- Distributor network support worldwide including certified distribution FAEs

Business Scope

Microchip Technology Inc. manufacturers and markets a variety of VLSI CMOS semiconductor components to support the market for cost-effective embedded control solutions. In particular, the company specializes in highly integrated, field-programmable RISC MCUs, application-specific standard products, secure data products, application-specific integrated circuits and related Serial EEPROM memory products to meet growing market requirements for high performance, yet economical embedded control capability in products. Microchip's products feature the industry's most economical One-Time-Programmable (OTP) EPROM, reprogrammable Flash and EEPROM, and ROM capability, along with the compact size, integrated functionality, ease of development and technical support so essential to timely and cost-effective product development by our customers.



Chandler, Arizona: Company headquarters near Phoenix, Arizona; executive offices, R&D and wafer fabrication occupy this 242,000-square-foot multi-building facility.



Tempe, Arizona: Microchip's 170,000-square-foot wafer fabrication facility provides increased manufacturing capacity today and for the future.

© 1997 Microchip Technology Inc.

Market Focus

Microchip targets selected markets where our advanced designs, progressive process technology and industry-leading product performance enable us to deliver decidedly superior performance. The company has positioned itself to maintain a dominant role as a supplier of high-performance, field-programmable OTP MCUs and associated memory for embedded control applications which are found throughout the consumer, automotive, telecommunication, office automation and industrial control markets. Microchip products are also meeting the unique design requirements of targeted embedded applications including security, battery management and field-programmable gate array conversions.

Certified Quality Systems

Microchip received ISO 9001 Quality System certification for its worldwide headquarters and wafer fabrication facilities in January 1997. The company's field-programmable 8-bit MCUs, Serial EEPROMs, related specialty memory products and development systems conform to the stringent quality standards of the International Standards Organization (ISO). The company's assembly and test facility in Kaohsiung, Taiwan is ISO 9002 certified. In addition, Microchip's manufacturing and engineering systems were audited to meet the Automotive Electronics Council A100 guidelines for overall quality, reliability and demonstrated conformance to the requirements of QS-9000 in 1996.

DNV Certification, Inc. USA
ACCREDITED REGISTRAR ANSI-RAB
DNV
DNV MSC The Netherlands Accredited by the RvA
ISO 9001 REGISTERED FIRM

Fully Integrated Manufacturing

Microchip delivers fast turnaround and consistent quality through total control over all phases of production. Research and development, design, mask making, wafer fabrication, and the major part of assembly and quality assurance testing are conducted at facilities wholly-owned and operated by Microchip. Our integrated approach to manufacturing along with rigorous use of advanced Statistical Process Control (SPC) and a continuous improvement culture has resulted in high and consistent yields which have positioned Microchip as a quality leader in its global markets. Microchip's unique approach to SPC provides customers with excellent costs, quality, reliability and on-time delivery.



Bangkok, Thailand: Microchip's 140,000-square-foot manufacturing facility houses the technology and assembly/test equipment for high speed testing and packaging.

A Global Network of Plants and Facilities

Microchip is a global competitor providing local service to the world's technology centers. The Company's design, technology advancement, front-end wafer fabrication, and wafer probe and sort facilities are located in Chandler and Tempe, Arizona.

The Tempe facility provides an additional 170,000 square feet of manufacturing space that meets the increased production requirements of a growing customer base, and provides production capacity which more than doubles that of Chandler. Assembly and test facilities, predominantly located in Kaohsiung, Taiwan and Bangkok, Thailand house the technology and assembly and test equipment necessary for modern plastic and ceramic packaging.

During fiscal 1996, Microchip invested more than $115 million for capital additions to its Tempe fabrication facility and for construction of the additional Bangkok testing facility in Asia. These investments have led to significant improvements in the Company's output and overall cost structure.

Sales and application offices are located in key cities throughout the Americas, Asia/Pacific, Japan and Europe. Offices are staffed to meet the high quality expectations of our customers, and can be accessed for technical and business support.

© 1997 Microchip Technology Inc.

Embedded Control Overview

Unlike "processor" applications such as personal computers and workstations, the computing or controlling elements of embedded control applications are buried inside the application. The user of the product is only concerned with the very top-level user interface such as keypads, displays and high-level commands. Very rarely does an end-user know (or care to know) the embedded controller inside (unlike the conscientious PC users, who are intimately familiar not only with the processor type, but also its clock speed, DMA capabilities and so on).

It is, however, most vital for designers of embedded control products to select the most suitable controller and companion devices. Embedded control products are found in all market segments: consumer, commercial, PC peripherals, telecommunications (including personal telecommunication products), automotive and industrial. Most often embedded control products must meet special requirements: cost effectiveness, low-power, small-footprint and a high level of system integration.

Typically, most embedded control systems are designed around a MCU which integrates on-chip program memory, data memory (RAM) and various peripheral functions, such as timers and serial communication. In addition, these systems usually require complementary Serial EEPROM, display drivers, keypads or small displays.

Microchip has established itself as a leading supplier of field-programmable embedded control solutions. The combination of high-performance MCU from the PIC12CXXX, PIC16C5X, PIC16CXXX and PIC17CXXX families, along with non-volatile memory products, provide the basis for this leadership.

Microchip is committed to continuous innovation and improvement in design, manufacturing and technical support to provide the best possible embedded control solutions to you.

PICmicro MCU Overview and Roadmap

Microchip PICmicro MCUs combine high performance, low cost, and small package size, offering the best price/performance ratio in the industry. More than 100 million of these devices ship each year to cost-sensitive consumer products, computer peripherals, office automation, automotive control systems, security and telecommunication applications.

Microchip offers four families of 8-bit MCUs to best fit your needs: PIC16C5X 12-bit program word, PIC16CXXX 14-bit program word, PIC17CXXX 16-bit program word and PIC12CXXX 8-pin 12-bit/14-bit program word MCU families.

All families offer OTP, low-voltage and low-power options, as well as various packaging options. Selected members are available in ROM and reprogrammable Flash versions.

The widely-accepted PIC16C5X, PIC16CXXX and PIC17CXXX families are the industry's only 8-bit MCUs using a high-speed RISC architecture. Today, these families are joined by the industry's first and only 8-pin MCU family – the PIC12CXXX. The PIC12CXXX family combines the 8-bit high-speed RISC architecture of the PICmicro MCUs with the smallest footprint MCU. Microchip pioneered the use of RISC architecture to obtain high speed and instruction efficiency.

PIC12CXXX: 8-Pin, 8-Bit Family

The PIC12CXXX family packs Microchip's powerful RISC-based PICmicro architecture into 8-pin DIP and SOIC packages. These PIC12CXXX products have 12-bit or 14-bit wide instruction sets, low operating voltage of 2.5V, small package footprint, interrupt handling and a deeper hardware stack. All of these features provide an intelligence level to applications not previously available because of cost or size.

PIC16C5X: 12-Bit Architecture Family

The PIC16C5X is the well established base-line family offering the most cost-effective solution. These PIC16C5X products have a 12-bit wide instruction set and are currently offered in 18-, 20- or 28-pin packages. In the SOIC and SSOP packaging options, these are among the smallest footprint MCUs. Low-voltage operation, down to 2.0V for OTPs, make this family ideal for battery operated applications.

PIC16CXXX: 14-Bit Architecture Family

The PIC16CXXX family offers a wide-range of options, from 18-pin to 68-pin packages as well as low to high levels of peripheral integration. This family has a 14-bit wide instruction set, interrupt handling capability and a deep 8-level hardware stack. The PIC16CXXX family provides the performance and versatility to meet the requirements of more demanding, yet cost-sensitive, mid-range 8-bit applications.

The PIC14C000 Programmable Mixed-Signal Controller allows engineers to design intelligent controllers for smart batteries, battery chargers, battery status monitoring, uninterruptible power supplies, HVAC, and other data acquisition and processing required for managing energy. The PIC14C000 can support any battery technology including Li Ion, NiMH, NiCd, Pb acid, Zinc Air. In addition, the product's I^2C™ port enables any system OEM, battery pack VAR, and battery manufacturer to design, build, and market SBD-compliant products supporting the System Management Bus standard.

PIC17CXXX: 16-Bit Architecture Family

The PIC17CXXX family offers the world's fastest execution performance of any 8-bit MCU family in the industry. The PIC17CXXX family extends the PICmicro MCUs high-performance RISC architecture with a 16-bit instruction word, enhanced instruction set and powerful vectored interrupt handling capabilities. A powerful array of precise on-chip peripheral features provide the performance for the most demanding 8-bit applications.

© 1997 Microchip Technology Inc.

PICmicro MCU Naming Convention

The PICmicro architecture offers users a wide range of cost/performance options of any 8-bit MCU family. In order to identify the families, the following naming conventions have been applied to the PICmicro MCUs.

TABLE 1: PICMICRO MCU NAMING CONVENTION*

Family		Architectural Features	Name	Technology	Products
PIC17CXXX	8-bit High-Performance MCU Family	• 16-bit wide instruction set • Internal/external vectored interrupts • DC - 25 MHz clock speed • 120 ns instruction cycle (@ 33 MHz) • Hardware multiply	PIC17C4X	OTP program memory, digital only	PIC17C42A, PIC17C43, PIC17C44
			PIC17CR4X	ROM program memory, digital only	PIC17CR42, PIC17CR43
			PIC17C75X	OTP program memory with mixed-signal functions	PIC17C756
PIC16CXXX	8-bit Mid-Range MCU Family	• 14-bit wide instruction set • Internal/external interrupts • DC - 20 MHz clock speed (Note 1) • 200 ns instruction cycle (@ 20 MHz)	PIC14CXXX	OTP program memory with A/D and D/A functions	PIC14C000
			PIC16C55X	OTP program memory, digital only	PIC16C554, PIC16C556, PIC16C558
			PIC16C6X	OTP program memory, digital only	PIC16C62, PIC16C62A, PIC16C63, PIC16C64, PIC16C64A, PIC16C65, PIC16C65A, PIC16C66, PIC16C67
			PIC16CR6X	ROM program memory, digital only	PIC16CR62, PIC16CR63, PIC16CR64, PIC16CR65
			PIC16C62X	OTP program memory with comparators	PIC16C620, PIC16C621, PIC16C622
			PIC16C7X	OTP program memory with analog functions (i.e. A/D)	PIC16C710, PIC16C71, PIC16C711, PIC16C715, PIC16C72, PIC16C73, PIC16C73A, PIC16C74, PIC16C74A, PIC16C76, PIC16C77
			PIC16F8X	Flash program and EEPROM data memory	PIC16C84 PIC16F83, PIC16F84
			PIC16CR8X	ROM program and EEPROM data memory	PIC16CR83, PIC16CR84
			PIC16C9XX	OTP program memory, LCD driver	PIC16C923, PIC16C924
PIC16C5X	8-bit Base-Line MCU Family	• 12-bit wide instruction set • DC - 20 MHz clock speed • 200 ns instruction cycle (@ 20 MHz)	PIC16C5X PIC16C5XA	OTP program memory, digital only	PIC16C52, PIC16C54, PIC16C54A, PIC16C55, PIC16C56, PIC16C57, PIC16C58A
			PIC16CR5X PIC16CR5XA	ROM program memory, digital only	PIC16CR54A, PIC16CR57B, PIC16CR58A
PIC12CXXX	8-bit, 8-pin MCU Family	• 12- or 14-bit wide instruction set • DC - 4 MHz clock speed • 1000 ns instruction cycle (@ 4 MHz)	PIC12C5XX	OTP program memory, digital only	PIC12C508, PIC12C509
			PIC12C67X	OTP program memory with analog functions	PIC12C671, PIC12C672

Note 1: The maximum clock speed for some devices is less than 20 MHz.

Please check with your local Microchip distributor, sales representative or sales office for the latest product information.

*As of April 1997.

© 1997 Microchip Technology Inc.

The Advantage of Field Programmability

The PICmicro MCU family provides a unique combination of a high-performance RISC processor with cost-effective OTP technology. Cost-effective OTP provides many benefits to the user at prices which can be comparable to competing ROM solutions. The benefits include:

1. Quick time-to-market.
2. Ease of code changes.
3. Ability to provide adaptable solutions to end-customer requirements.
4. Ability to meet upside potential via inventory positions at Microchip or worldwide distribution.
5. Reduced scrappage in manufacturing.
6. Reduced inventory in manufacturing.
7. Reduced work-in-process liability.

For most manufacturers, getting the product to market quickly has become the number one goal as global markets have become more competitive. Time-to-market puts pressure on all functions within the manufacturing process: development, purchasing, production, marketing and sales. Field-programmable OTP technology streamlines the process for all stages in the product life cycle. In the early product development stages, a programmable MCU allows much of the functionality to be implemented in software which can be modified more easily than hardware-only solutions.

In the manufacturing stage, the compression of the product life cycle curve puts pressure on the management of inventory and manufacturing cycle times. Minimizing inventory reduces the ability to meet upside demand. Using a traditional ROM-based MCU limits the ability to respond to the market with product enhancements or semi-customized products for specific customers. Using the standard OTP-based PICmicro MCU solves all these issues. Inventory can be managed effectively by using the same device in several systems. Costs can be reduced due to volume purchasing. Upsides can be met from either safety stock, directly from Microchip, or local distributors who regularly inventory all PICmicro MCU devices. A sudden decline in demand means no work-in-process ROM-based inventory and any excess safety stock can be consumed by the other products using the same standard device.

OTP is the 'Flexible Manufacturing' technology of the MCU world. As competition intensifies, the demand for customer-specific products increases. Having the ability to change (for example, the appearance of LCD displays or add extra features in a timely manner) can be a key competitive advantage. Programming the OTP device on the manufacturing floor allows easy customizing and internal tracking of the devices for each specific customer. Customization can significantly increase the overall product life cycle to provide better return on investment and help minimize the threat of competition.

Current PICmicro MCU product families include advanced features such as sophisticated timers, embedded analog-to-digital converters, extended instruction/data memory, inter-processor communication (I^2C bus, SPI™ and USARTs) and ROM, RAM, EPROM, EEPROM and Flash memories.

All PICmicro families are supported by user-friendly development systems including; assembler, software simulator, C Compiler, fuzzy logic development software, programmers and in-circuit emulators.

© 1997 Microchip Technology Inc.

Development Systems

Microchip is committed to providing useful and innovative solutions to your embedded system designs. Our installed base of application development systems has grown to an impressive 90,000 systems worldwide. Significantly, more than half of the total system shipments over the past six years took place within the past 24 months.

Among support products offered are PICMASTER® Real-Time Universal In-circuit Emulator running under the Windows® environment. MPLAB™, a complete Integrated Development Environment (IDE), is provided with PICMASTER. MPLAB allows the user to edit, compile and emulate from a single user interface, making the developer productive very quickly. PICMASTER is designed to provide product development engineers with an optimized design tool for developing target applications. This universal in-circuit emulator provides a complete MCU design tool set for all MCUs in the PIC12CXXX, PIC16C5X, PIC16CXXX and PIC17CXXX families. A CE compliant version of PICMASTER is available for European Union (EU) countries.

ICEPIC is a low-cost in-circuit emulator solution for the Microchip PIC16C5X and PIC16CXXX MCU families. PRO MATE® II, the full-featured, modular device programmer, enables you to quickly and easily program user software into PICmicro MCUs. PRO MATE II runs under MPLAB IDE and operates as a stand-alone unit or in conjunction with a PC-compatible host system. The PICSTART® Plus and Lite development kits, are low-cost development systems for the PIC12CXXX, PIC16C5X/ PIC16CXXX and PIC17CXXX MCUs. PICDEM low-cost demonstration boards are simple boards which demonstrate the basic capabilities of the full range of Microchip's MCUs. Users can program the sample MCUs provided with PICDEM boards, on a PRO MATE II or PICSTART programmer, and easily test firmware. KEELOQ Evaluation and Programming Tools support Microchip's HCS Secure Data Products.

The Serial EEPROM Designer's Kit includes everything necessary to read, write, erase or program special features of any Microchip Serial EEPROM product including *Smart Serials*™ and secure serials. The *Total Endurance*™ Disk is included to aid in trade-off analysis and reliability calculations. The total kit can significantly reduce time-to-market and result in an optimized system.

TABLE 2: PICMICRO SYNERGISTIC DEVELOPMENT TOOLS

Development Tool	Name	PIC12CXXX	PIC16C5X	PIC16CXXX	PIC17CXXX
Integrated Development Environment (IDE)	MPLAB™	✔	✔	✔	✔
Universal Macro-Assembler	MPASM	✔	✔	✔	✔
C Compiler	MPLAB-C	✔	✔	✔	✔
Full-Featured, Modular In-Circuit Emulator	PICMASTER®	✔	✔	✔	✔
Low-Cost Modular In-Circuit Emulator	ICEPIC	—	✔	✔	—
Full-Featured, Modular Device Programmer	PRO MATE® II	✔	✔	✔	✔
Entry-Level Development Kit with Programmer	PICSTART® Plus	✔	✔	✔	✔
Low-Cost Entry Level Development Kit	PICSTART® Lite	✔	✔	✔	Planned
Fuzzy Logic Development Software	*fuzzy*TECH®-MP	✔	✔	✔	✔
Application Code Generator	MP-DriveWay™	—	✔	✔	✔

© 1997 Microchip Technology Inc.

Software Support

MPLAB is a Windows-based development platform for Microchip's PICmicro OTP MCUs. MPLAB IDE offers a project manager and program text editor, a user-configurable toolbar containing four pre-defined sets and a status bar which communicates editing and debugging information.

MPLAB is the common user interface for Microchip development systems tools including MPLAB Editor, MPASM Assembler, MPLAB-SIM Software Simulator, PICMASTER In-Circuit Emulator, PRO MATE II Programmer and PICSTART Plus Development Programmer.

Microchip's PICmicro MCUs are also supported by fuzzy logic development software, an application code generator and a C-Compiler. MP-DriveWay is an easy-to-use Windows-based Application Code Generator for visually configuring all the peripherals in a PICmicro device and, with a click of the mouse, generate all the initialization and many functional code modules in C language.

Microchip endeavors at all times to provide the best service and responsiveness possible to its customers. The Microchip Internet Home Page can provide you with the latest technical information, production released software for development tools, application notes and promotional news on Microchip products and technology. The Microchip World Wide Web address is http://www.microchip.com.

The Microchip Systems Bulletin Board Service (BBS) is one service to facilitate this service. It is a multi-faceted tool that can provide you with information on a number of different topics.

Special Interest Groups (SIGs) available through the BBS can provide you with the opportunity to discuss issues and topics of interest with others that share your interest or questions. The BBS is regularly used to distribute technical information, application notes, source code, errata sheets, bug reports, interim patches for Microchip systems products, and user contributed files for distribution.

Please see the "On-Line Technical Support" Section for more information on the Microchip BBS and WWW site.

Application-Specific Standard Products (ASSPs)

Microchip ASSPs provide value-added embedded control solutions by combining PICmicro MCU architecture, non-volatile memory, and innovative software technology for vertical applications. These products incorporate technology that offers a complete solution that is both unique to the customer and standard in manufacture to Microchip. In addition, Microchip ASSPs reduce or remove the barriers for customers to use Microchip solutions, in their products, through the use of software, embedded in secure OTP- or ROM-based MCUs. These MCUs are packaged to provide the highest integration, to the customer, at the best overall system cost.

The MTE1122 Energy Management Controller combines Microchip's proprietary PICmicro 8-bit RISC MCU technology with a unique, patent pending power management firmware algorithm in a single package. This device, by monitoring and controlling the supply requirements into an AC induction motor, effectively reduces the power consumed by the motor. The MTE1122 is available in both plastic DIP and space-saving SOIC packages, and operates over commercial and industrial temperature ranges.

Future ASSP products will include advanced features such as mixed analog and digital capability as well as an ever broadening family of turnkey software solutions for the embedded control market.

Application-Specific Integrated Circuits (ASICs)

Microchip offers the QuickASIC™ Mask Gate Array (MPGA) family which replaces standard field-programmable gate arrays (FPGAs) and complex programmable logic devices (CPLDs) with a MPGA at a substantially reduced price per unit compared to programmable logic devices. Microchip's QuickASIC family offers turnkey digital design conversion of most FPGA/CPLD devices by transforming its net list into a masked gate array – with very fast lead times of four weeks from design signoff to prototype. QuickASIC devices can be used for any application prototyping with FPGAs/CPLDs including multimedia, data communications, telecommunications and computer peripherals.

In addition, Microchip plans call for customers being given the opportunity to combine Microchip's powerful RISC-based PICmicro MCU core with configurable gate arrays, providing a much wider range of flexibility, power ranges and custom functionality.

© 1997 Microchip Technology Inc.

Serial EEPROM Overview

Microchip offers one of the broadest selections of CMOS Serial EEPROMs on the market for embedded control systems. Serial EEPROMs are available in a variety of densities, operating voltages, bus interface protocols, operating temperature ranges and space saving packages.

Densities:

Currently range from 128 bits to 64K bits with higher density devices in development.

Bus Interface Protocols:

All major protocols are covered: I^2C, Microwire® and SPI.

Operating Voltages:

In addition to standard 5V devices there are two low voltage families. The "LC" devices operate down to 2.5V, while the breakthrough "AA" family operates, in both read and write mode, down to 1.8V, making these devices highly suitable for alkaline and NiCd battery powered applications.

Temperature Ranges:

Like all Microchip devices, Serial EEPROMs are offered in Commercial (0°C to +70°C), Industrial (-40°C to +85°C) and Extended (-40°C to +125°C) operating temperature ranges.

Packages:

Small footprint packages include: industry standard 5-lead SOT-23, 8-lead DIP, 8-lead SOIC in JEDEC and EIAJ body widths, and 14-lead SOIC. The SOIC comes in two body widths; 150 mil and 207 mil.

Technology Leadership:

Microchip's Serial EEPROMs are backed by a 10 million Erase/Write cycle guarantee. Microchip's erase/write cycle endurance is among the best in the world, and only Microchip offers such unique and powerful development tools as the Total Endurance disk. This mathematical software model is an innovative tool used by system designers to optimize Serial EEPROM performance and reliability within the application.

The Company has also developed the world's first 64K Smart Serial EEPROM. Device densities range from 128 bits up to 64K bits. Another first is the 24LC21, a single chip DDC1/DDC2™-compatible solution for plug-and-play video monitors. In addition, Microchip released a high-speed 1 MHz 2-wire Serial EEPROM device ideal for high-performance embedded systems.

Microchip is a high-volume supplier of Serial EEPROMs to all the major markets worldwide including consumer, automotive, industrial, computer and communications. To date, more than 700 million units have been produced. Microchip continues to develop new Serial EEPROM solutions for embedded control applications.

Parallel EEPROM Overview

CMOS Parallel EEPROM devices from Microchip are available in 4K, 16K and 64K densities. The manufacturing process used for these EEPROMs ensures 10,000 to 100,000 erase/writecycles typical. Data retention is more than 10 years. Fast write times are less than 200 µs. These EEPROMs work reliably under demanding conditions and operate efficiently at temperatures from -40°C to +85°C. Microchip's expertise in advanced SOIC, TSOP and VSOP surface mount packaging supports our customers' needs in space-sensitive applications.

Typical applications include computer peripherals, engine control, telecommunications and pattern recognition.

OTP EPROM Overview

Microchip's CMOS EPROM devices are produced in densities from 64K to 512K. Typical applications include computer peripherals, instrumentation, and automotive devices. Microchip's expertise in surface mount Packaging on SOIC, TSOP and VSOP packages led to the development of the Surface Mount OTP EPROM market where Microchip is a leading supplier today. Microchip is also a leading supplier of low-voltage EPROMs for battery powered applications.

Secure Data Products Overview

Microchip's patented KEELOQ code hopping technology is the perfect solution for remote keyless entry and logical/physical access control systems. The initial device in the family, HCS300 encoder, will replace current fixed code encoders in transmitter applications and provide a low cost, integrated solution. The KEELOQ family is rapidly expanding with the HCS301 (high voltage encoder), HCS200 (low-end, low-cost encoder), and high-end encoders (HCS360 and HCS361) that meet OEM specifications and requirements.

Microchip provides flexible decoder solutions by providing optimized routines for Microchip's PICmicro MCUs. This allows the designer to combine the decoder and system functionality in a MCU. The decoder routines are available under a license agreement. The HCS509 and HCS512 are the first decoder devices in the KEELOQ family. These devices are single chip decoder solutions and simplify designs by handling learning and decoding of transmitters.

The KEELOQ product family is expanding to include enhanced encoders and decoders. Typical applications include automotive RKE, alarm and immobilizer systems, garage door openers and home security systems.

KEELOQ™
Rock Solid Security

© 1997 Microchip Technology Inc.

Ease of Production Utilizing Quick Turn Programming (QTP) and Serialized Quick Turn Programming (SQTP SM)

Recognizing the needs of high-volume manufacturing operations, Microchip has developed two programming methodologies which make the OTP products as easy to use in manufacturing as they are efficient in the system development stage.

Quick Turn Programming allows factory programming of OTP products prior to delivery to the system manufacturing operation. PICmicro, EPROM and Serial EEPROM products can be automatically programmed, with the users program, during the final stages of the test operation at Microchip's assembly and test operations in the Philippine Islands, Taiwan and Thailand. This low-cost programming step allows the elimination of programming during system manufacturing and essentially allows the user to treat the PICmicro and memory products as custom ROM products. With one- to four-week lead times on QTP products, the user no longer needs to plan for the extended ROM masking lead times and masking charges associated with custom ROM products. This capability, combined with the off-the-shelf availability of standard OTP product, ensures the user of product availability and the ability to reduce his time-to-market once product development has been completed.

Unique in the 8-bit MCU market is Microchip's ability to enhance the QTP capability with Serialized Quick Turn Programming (SQTP). SQTP allows for the programming of devices with unique, random or serialized identification codes. As each PICmicro device is programmed with the customers program code, a portion of the program memory space can be programmed with a unique ID, accessible from normal program memory, which will allow the user to provide each device with a unique identification. This capability is ideal for embedded systems applications where the transmission of key codes or identification of the device as a node within a network is essential. Taking advantage of this capability allows the system designer to eliminate the requirement for expensive off-chip code implementation using DIP switches or non-volatile memory components. The SQTP offering, pioneered by Microchip, provides the embedded systems designer with a low cost means of putting a unique and custom device into every system or node.

Future Products and Technology

Microchip is constantly developing advanced process technology modules and new products that utilize the Company's advanced manufacturing capabilities. Current production technology utilizes lithography dimensions down to 0.7 micron.

Microchip's research and development activities include exploring new process technologies and products that have industry leadership potential. Particular emphasis is placed on products that can be put to work in high-performance broad-based markets.

Equipment is continually updated to bring the most sophisticated process, CAD and testing tools online. Cycle times for new technology development are continuously reduced by using in-house mask generation, a high-speed pilot line within the manufacturing facility and continuously improving methodologies.

More advanced technologies are under development, as well as advanced CMOS RISC-based MCU, ASSP and CMOS EEPROM and EPROM, QuickASIC and other products. Objective specifications for new products are developed by listening to our customers and by close co-operation with our many customer-partners worldwide.

© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

SECTION 2 PIC16C5X APPLICATION NOTES

2

Implementation of an Asynchronous Serial I/O - AN510 2-1
PLD Replacement - AN511 2-19
Implementing Ohmmeter/Temperature Sensor - AN512 2-41
Analog to Digital Conversion Using a PIC16C54 - AN513 2-47
Software Interrupt Techniques - AN514 2-53
Implementing a Simple Serial Mouse Controller - AN519 2-57
A Comparison of 8-Bit Microcontrollers - AN520 2-69
Interfacing to AC Power Lines - AN521 2-79
Power-up Considerations - AN522 2-81
Software Stack Management - AN527 2-85
Implementing Wake-up on Key Stroke - AN528 2-89
Multiplexing LED Drive and a 4x4 Keypad Sampling - AN529 2-95
Intelligent Remote Positioner (Motor Control) - AN531 2-121
Using a PIC16C5X as a Smart I^2C^{TM} Peripheral - AN541 2-135
Using PIC16C5X Microcontrollers as LCD Drivers - AN563 2-151
PIC16C54A EMI Results - AN577 2-165
Implementing Long Calls - AN581 2-171
Macros for Page and Bank Switching - AN586 2-175
A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs - AN590 2-197
Frequency Counter Using PIC16C5X - AN592 2-209
Serial Port Routines Without Using Timer0 - AN593 2-219
Modifying PIC16C54A Code for the PIC16C58A - AN618 2-225
Code Development for the PIC16C52 - AN641 2-233
D/A Conversion Using PWM and R-2R Ladders to Generate Sine and DTMF Waveforms - AN655 2-235
Decoding Infrared Remote Controls Using a PIC16C5X Microcontroller - AN657 2-253

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

Implementation of an Asynchronous Serial I/O

Author: *Amar Palacherla*
Microchip Technology Inc.

INTRODUCTION

The PIC16C5X series from Microchip Technology Inc., are 8-bit, high-speed, EPROM-based microcontrollers. This application note describes an implementation of an asynchronous serial I/O using a PIC16C5X microcontroller. These microcontrollers can operate at very high speeds with a minimum cycle time of 200 ns @ 20 MHz input clock. Many microcontroller applications require chip-to-chip serial data communications. Since the PIC16C5X series has no on-chip serial ports, serial communication has to be performed in software. For many cost-sensitive high volume applications, implementation of a serial I/O through software provides a more cost effective solution than dedicated logic. This application note provides code for the PIC16C5X to simulate a serial port using two I/O pins (one as input for reception and the other as output for transmission).

IMPLEMENTATION

Two programs are provided in this application note. One program (Appendix B) simulates full duplex RS-232 communication and the other (Appendix A) provides implementation of half-duplex communication. Using half-duplex, baud rates up to 19200 can be implemented using an 8 MHz input clock. For full-duplex, the software can handle up to 9600 baud at 8 MHz and 19200 baud at 20 MHz, one or two stop bits, eight or seven data bits, no Parity and can transmit or receive with either LSb first (normal mode) or MSb first (CODEC-like mode). It should be noted that the higher the input clock the better the resolution. These options should be set up during assembly time and not during run time. The user simply has to change the header file for the required communication options. The software does not provide any handshaking protocols. With minor modifications, the user may incorporate software handshaking using XON/XOFF. To implement hardware handshaking, an additional two digital I/O pins may be used as RTS (ready to send) and CTS (clear to send) lines.

© 1997 Microchip Technology Inc.

Figure 1 shows a flowchart for serial transmission and Figure 2 shows a flowchart for reception. The flowcharts show case transmission/reception with LSb first and eight data bits. For reception, the data receive pin, DR, is polled approximately every B/2 seconds (52 µs for 9600 baud) to detect the start bit, where B is the time duration of one bit (B = 1/Baud). If a start bit is found, then the first data bit is checked for after 1.25B seconds. From then on, the other data bits are checked every B seconds (104 µs for 9600 baud).

In the case of transmission, first a start bit is sent by setting the transmit data pin, DX, to zero for B seconds, and from then on the DX pin is set/cleared corresponding to the data bit every B seconds. Assembly language code corresponding to the following flowcharts is given in Example 1 and Example 2.

FIGURE 1: TRANSMISSION FLOWCHART

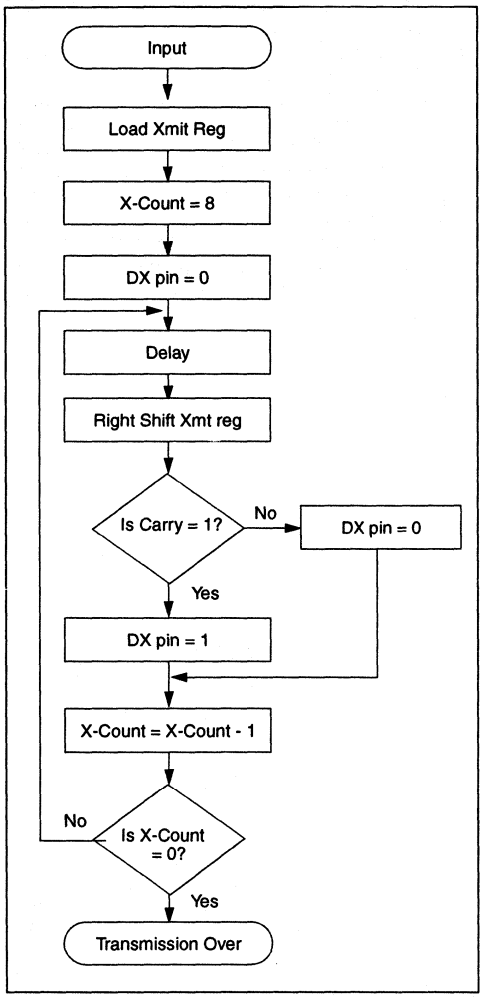
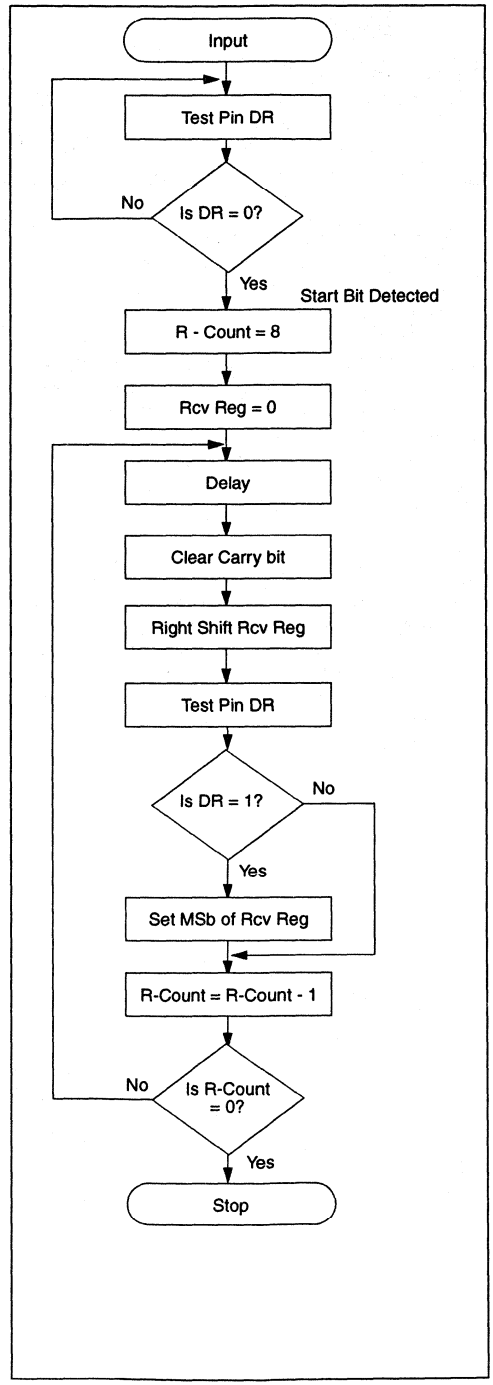


FIGURE 2: RECEPTION FLOWCHART



© 1997 Microchip Technology Inc.

EXAMPLE 1: TRANSMIT ASSEMBLY CODE (CORRESPONDING TO FIGURE 1)

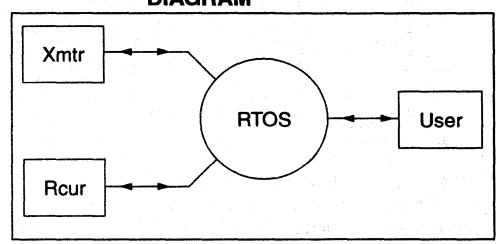
```
;********************* Transmitter*****************************
Xmtr    movlw   8             ; Assume XmtReg contains data to be Xmted
        movwf   XCount        ; 8 data bits
        bcf     Port_A,DX     ; Send Start Bit
X_next  call    Delay         ; Delay for B/2 Seconds
        rrf     XmtReg
        btfsc   STATUS,CARRY  ; Test the bit to be transmitted
        bsf     Port_A,DX     ; Bit is a one
        btfss   STATUS,CARRY
        bcf     Port_A,DX     ; Bit is zero
        decfsz  Count         ; If count = 0, then transmit a stop bit
        goto    X_next        ; transmit next bit
;
X_Stop  call    Delay
        bsf     Port_A,DX     ; Send Stop Bit
X_Over  goto    X_Over
```

EXAMPLE 2: RECEIVE ASSEMBLY CODE (CORRESPONDING TO FIGURE 2)

```
;****************    Receiver  *********************
        ;
        Rcvr    btfsc   Port_A,DR       ; Test for Start Bit
                goto    Rcvr            ; Start Bit not found
                movlw   8               ; Start Bit Detected
                movwf   RCount          ; 8 Data Bits
                clrf    RcvReg          ; Receive Data Register
        R_next  call    Delay           ; Delay for B/2 Seconds, B=Time duration of
                                        ;  1bit
                bcf     STATUS,CARRY    ; Clear CARRY bit
                rrf     RcvReg          ; to set if MSB first or LSB first
                btfsc   Port_A,DR       ; Is the bit a zero or one ?
                bsf     RcvReg,MSB      ; Bit is a one
                call    Delay
                decfsz  RCount
                goto    R_next
        R_Over  goto    R_Over          ; Reception done
```

The software is organized such that the communication software acts as a Real-Time Operating System (RTOS) which gives control to the user routine for a certain time interval. After this predetermined time slot, the user must give back control to the Operating System. This is true only in the case of full-duplex implementation. Timing considerations are such that the user gets control for approximately half the time of the bit rate and the rest of the time is used up by the Operating System (and software delays). Please refer to Table 1 for the delay constants and the time the user gets using an 8 MHz input clock. Delay constants and the time that the user gets at 20 MHz and 4 MHz input clock speeds are given in the source code listing of the full-duplex routine. At frequencies other than 4, 8, or 20 MHz, the delay constants and the time the user gets can be computed from the equations given in Equation 1.

FIGURE 3: FULL-DUPLEX BLOCK DIAGRAM



EQUATION 1: EQUATIONS FOR DELAY CONSTANTS

```
Note: CLKOUT = Fosc/4
Baud_Cycles = Clkout/Baud;
User_time = Baud_Cycles • (float) 0.5;
K0 = (1.25 • Baud_Cycles - 2.0 • User_time - 89)/3.0; IF (K0 < 0)
   {
   K0 = 0.0;
   User_time = 0.50 • (1.25 • Baud_Cycles - 89.0) ;
   }
K1 = (1.25 • Baud_Cycles  - User_time - 59.0 - 3 • K0)/3.0 ;
K2 = (Baud_Cycles - User_time - 41.0 - 3 • K0)/3.0 ;
K3 = (Baud_Cycles - User_time - 61.0 - 3 • K0)/3.0  ;
K4 = (Baud_Cycles - User_time - 55.0 - 3 • K0)/3.0  ;
K5 = (Baud_Cycles - User_time - 55.0 - 3 • K0)/3.0 +1.0 ;
K6 = 0.0;
K7 = (1.25 • Baud_Cycles - User_time - 39.0 - 3 • K0)/3.0 ;
```

TABLE 1: DELAY CONSTANTS AT 8 MHz INPUT CLOCK

Constant	19200	9600	4800	2400	1200
K0	-	0	5	39	109
K1	-	39	80	150	288
K2	-	27	51	86	155
K3	-	21	44	80	148
K4	-	23	46	82	150
K5	-	24	47	83	151
K6	-	0	0	0	0
K7	-	45	86	156	295
User Cycles	-	86	208	416	832

TABLE 2: DELAY CONSTANTS AT 20 MHz INPUT CLOCK

Constant	19200	9600	4800	2400	1200
K0	0	13	57	143	317
K1	49	98	184	358	705
K2	34	60	103	191	364
K3	27	53	96	184	357
K4	29	55	98	186	359
K5	30	56	99	187	360
K6	0	0	0	0	0
K7	56	104	190	365	712
User Cycles	118	260	521	1042	2083

For example, if the baud rate selected is 9600 bps (@ 8 MHz), then the total time frame for one bit is approximately 104 µs. Out of this 104 µs, 61 µs are used by the Operating System and the other 43 µs are available to the user. It is the user's responsibility to return control to the Operating System exactly after the time specified in Table 1. For very accurate timing (with resolution up to one clock cycle) the user may set up Timer0 with the Prescaler option for calculating the real-time. With TMR0 configured to increment on internal CLKOUT (500 ns @ 8 MHz CLKIN) and the prescaler assigned to it, very accurate and long timing delay loops may be assigned. This method of attaining accurate delay loops is not used in the RS-232 code (RTOS), so that Timer0 is available to the user for other important functions. If Timer0 is not used for other functions, the user may modify the code to replace the software delay loops by counting TMR0. For an example of using TMR0 counting for exact timing delays, refer to the "user" routine in Full Duplex code (Appendix B).

The software uses minimal processor resources. Only six data RAM locations (File Registers) are used. The RTOS uses one level of stack, but it is freed once control is given back to the user. The Watchdog Timer (WDT) and Timer0 are not used. The user should clear the WDT at regular intervals, if the WDT is enabled.

The usage of the program is described in the following sections. The user should branch to location "Op_Sys" exactly after the time specified in Table 1 or as computed from equations in Equation 1.

Whereas, the transmission is totally under user control, the Reception is under the control of the Operating System. As long as the user does not set the X_flag, no transmission occurs. On the other hand the Operating System is constantly looking for a start bit and the user should not modify the R_done flag or the RcvReg.

TRANSMISSION

Transmit Data is output on the DX pin (bit0 of PORTA). In the user routine, the user should load the data to be transmitted in the XmtReg and set the X_flag (`bsf kwn FlagRX,X_flag`). This flag gets cleared after the transmission. The user should check this flag (X_flag) to see if a transmission is in progress. Modifying XmtReg when the X_flag is set will cause erroneous data to be transmitted.

RECEPTION

Data is received on pin DR (bit1 of PORTA). The user should constantly check the "R_done" flag to see if the reception is over. If a reception is in progress, the R_flag is set. If the reception is over, the "R_done" flag is set. The "R_done" flag gets cleared when the next start bit is detected. The user should constantly check the R_done flag, and if set, then the received word is in Register "RcvReg". This register gets cleared when a new start bit is detected. It is recommended that the RcvReg, be copied to another register after the R_done flag is set. The R_done flag also gets cleared when the next start bit is detected.

The user may modify the code to implement an N deep buffer (limited to the number of Data RAM locations available) for receive. Also, if receiving at high speeds, and if the N deep buffer is full, an XOFF signal (HEX 13) may be transmitted. When ready to receive more data, an XON signal (HEX 11) should be transmitted.

© 1997 Microchip Technology Inc.

SUMMARY

The PIC16C5X family of microcontrollers allow users to implement half or full duplex RS-232 communication in software.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: ASSEMBLY LANGUAGE FOR HALF DUPLEX

```
MPASM 01.40 Released          HALF_DUP.ASM   1-16-1997  11:48:17          PAGE  1


LOC  OBJECT CODE  LINE SOURCE TEXT
  VALUE

                  00001          LIST    P = 16C54, n = 66
                  00002 ;
                  00003 ;*****************************************************************
                  00004 ;                        RS-232 Communication With PIC16C54
                  00005 ;
                  00006 ;         Half Duplex Asynchronous Communication
                  00007 ;
                  00008 ;         This program has been tested at Bauds from 1200 to 19200 Baud
                  00009 ;         ( @ 8,16,20 Mhz CLKIN )
                  00010 ;
                  00011 ;       As a test, this program will echo back the data that has been
                  00012 ;       received.
                  00013 ;
                  00014 ;       Program:          HALF_DUP.ASM
                  00015 ;       Revision Date:
                  00016 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                  00017 ;
                  00018 ;*****************************************************************
                  00019 ;
                  00020          INCLUDE         <P16C5X.INC>
                  00001          LIST
                  00002 ; P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology, Inc.
                  00224          LIST
                  00021
000001FF          00022 PIC54    equ 1FFH  ; Define Reset Vector
00000001          00023 Same     equ 1
00000007          00024 MSB      equ     7
                  00025
                  00026 ;****************  Communication Parameters   **********************
                  00027 ;
00000001          00028 X_MODE   equ 1 ; If ( X_MODE==1) Then transmit LSB first
                  00029                ; if ( X_MODE==0) Then transmit MSB first (CODEC like)
00000001          00030 R_MODE   equ 1 ; If ( R_MODE==1) Then receive LSB first
                  00031                ; if ( X_MODE==0) Then receive MSB first (CODEC like)
00000001          00032 X_Nbit   equ 1 ; if (X_Nbit==1) # of data bits (Transmission is 8 else 7
00000001          00033 R_Nbit   equ 1 ; if (R_Nbit==1) # of data bits (Reception) is 8 else 7
                  00034 ;
00000000          00035 Sbit2    equ 0 ; if Sbit2 = 0 then 1 Stop Bit else 2 Stop Bits
                  00036 ;
                  00037 ;*****************************************************************
00000005          00038 X_flag   equ PA0 ; Bit 5 of F3 ( PA0 )
00000006          00039 R_flag   equ PA1 ; Bit 6 of F3 ( PA1 )
                  00040 ;
00000000          00041 DX       equ 0   ; Transmit Pin ( Bit 0 of Port A )
00000001          00042 DR       equ 1   ; Reciive Pin ( Bit 1 of Port A )
                  00043 ;
                  00044 ;
00000044          00045 BAUD_1   equ .68  ; 3+3X = CLKOUT/Baud
00000043          00046 BAUD_2   equ .67  ; 6+3X = CLKOUT/Baud
00000022          00047 BAUD_3   equ .34  ; 3+3X = 0.5*CLKOUT/Baud
00000056          00048 BAUD_4   equ .86  ; 3+3X = 1.25*CLKOUT/Baud
00000042          00049 BAUD_X   equ .66  ; 11+3X = CLKOUT/Baud
00000042          00050 BAUD_Y   equ .66  ; 9 +3X = CLKOUT/Baud
```


© 1997 Microchip Technology Inc.

```
                    00051 ;
                    00052 ;*********************** Data RAM Assignments *********************
                    00053 ;
0008                00054         ORG  08H    ; Dummy Origin
                    00055 ;
0008                00056 RcvReg  RES  1      ; Data received
0009                00057 XmtReg  RES  1      ; Data to be transmitted
000A                00058 Count   RES  1      ; Counter for #of Bits Transmitted
000B                00059 DlyCnt  RES  1
                    00060 ;*****************************************************************
                    00061 ;
0000                00062         ORG     0
                    00063 ;
0000 0068           00064 Talk    clrf    RcvReg          ; Clear all bits of RcvReg
0001 0625           00065         btfsc   PORTA,DR        ; check for a Start Bit
0002 0A30           00066         goto    User            ; delay for 104/2 uS
0003 0923           00067         call    Delay4          ; delay for 104+104/4
                    00068 ;*****************************************************************
                    00069 ;       Receiver
                    00070 ;
0004                00071 Rcvr
                    00072         IF      R_Nbit
0004 0C08           00073         movlw   8               ; 8 Data bits
                    00074         ELSE
                    00075         movlw   7               ; 7 data bits
                    00076         ENDIF
                    00077 ;
0005 002A           00078         movwf   Count
0006 0403           00079 R_next  bcf     STATUS,C
                    00080         IF      R_MODE
0007 0328           00081         rrf     RcvReg,Same     ; to set if MSB first or LSB first
                    00082         ELSE
                    00083         rlf     RcvReg,Same
                    00084         ENDIF
0008 0625           00085         btfsc   PORTA,DR
                    00086 ;
                    00087         IF      R_MODE
                    00088           IF      R_Nbit
0009 05E8           00089           bsf     RcvReg,MSB      ; Conditional Assembly
                    00090           ELSE
                    00091           bsf     RcvReg,MSB-1
                    00092           ENDIF
                    00093         ELSE
                    00094         bsf     RcvReg,LSB
                    00095         ENDIF
                    00096 ;
000A 091F           00097         call    DelayY
000B 02EA           00098         decfsz  Count,Same
000C 0A06           00099         goto    R_next
                    00100 ;************************************************
000D 0208           00101 R_over  movf    RcvReg,0        ; Send back What is Just Received
000E 0029           00102         movwf   XmtReg
                    00103 ;************************************************
                    00104 ;       Transmitter
                    00105 ;
000F                00106 Xmtr
                    00107         IF      X_Nbit
000F 0C08           00108         movlw   8
                    00109         ELSE
                    00110         movlw   7
                    00111         ENDIF
0010 002A           00112         movwf   Count
                    00113 ;
                    00114         IF      X_MODE
                    00115         ELSE
                    00116           IF    X_Nbit
```


© 1997 Microchip Technology Inc.

```
                    00117           ELSE
                    00118           rlf     XmtReg,Same
                    00119           ENDIF
                    00120         ENDIF
                    00121 ;
0011 0405           00122         bcf     PORTA,DX        ; Send Start Bit
0012 0925           00123         call    Delay1
0013 0403           00124 X_next  bcf     STATUS,C
                    00125 ;
                    00126         IF      X_MODE
0014 0329           00127         rrf     XmtReg,Same      ; Conditional Assembly
                    00128         ELSE                     ; to set if MSB first or LSB first
                    00129         rlf     XmtReg,Same
                    00130         ENDIF
                    00131 ;
0015 0603           00132         btfsc   STATUS,C
0016 0505           00133         bsf     PORTA,DX
0017 0703           00134         btfss   STATUS,C
0018 0405           00135         bcf     PORTA,DX
0019 0921           00136         call    DelayX
001A 02EA           00137         decfsz  Count,Same
001B 0A13           00138         goto    X_next
001C 0505           00139         bsf     PORTA,DX        ; Send Stop Bit
001D 0925           00140         call    Delay1
                    00141 ;
                    00142         IF      Sbit2
                    00143         bsf     PORTA,DX
                    00144         call    Delay1
                    00145         ENDIF
                    00146 ;
001E 0A00           00147         goto    Talk            ; Back To Reception & Transmision
                    00148 ;
                    00149 ;   End of Transmission
                    00150 ;
001F 0C42           00151 DelayY  movlw   BAUD_Y
0020 0A28           00152         goto    save
0021 0C42           00153 DelayX  movlw   BAUD_X
0022 0A28           00154         goto    save
0023 0C56           00155 Delay4  movlw   BAUD_4
0024 0A28           00156         goto    save
0025 0C44           00157 Delay1  movlw   BAUD_1          ; 104 uS for 9600 baud
0026 0A28           00158         goto    save
0027 0C43           00159 Delay2  movlw   BAUD_2
0028 002B           00160 save    movwf   DlyCnt
0029 02EB           00161 redo_1  decfsz  DlyCnt,Same
002A 0A29           00162         goto    redo_1
002B 0800           00163         retlw   0
                    00164 ;
002C 0C0E           00165 main    movlw   0EH             ; Bit 0 of Port A is Output
002D 0005           00166         tris    PORTA           ; Set PORTA.0 as output ( DX )
002E 0525           00167         bsf     PORTA,DR
                    00168 ;
002F 0A00           00169         goto    Talk
                    00170 ;
                    00171 ;
0030 0C22           00172 User    movlw   BAUD_3
0031 002B           00173         movwf   DlyCnt
0032 02EB           00174 redo_2  decfsz  DlyCnt,Same
0033 0A32           00175         goto    redo_2
0034 0A00           00176         goto    Talk            ; Loop Until Start Bit Found
                    00177 ;
                    00178 ;
01FF                00179         ORG     PIC54
01FF 0A2C           00180         goto    main
                    00181 ;
                    00182         END
```


© 1997 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXX-----------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:     54
Program Memory Words Free:    458


Errors   :      0
Warnings :      0 reported,      0 suppressed
Messages :      0 reported,      0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: ASSEMBLY LANGUAGE LISTING FOR FULL DUPLEX

```
MPASM 01.40 Released            RS232.ASM   1-16-1997  12:12:09         PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004         TITLE   "RS232 Communication Using PIC16C54"
                     00005 ;
                     00006 ;    Comments :
                     00007 ;               (1) Full Duplex
                     00008 ;               (2) Tested from 1200 to 9600 Baud( @ 8 Mhz )
                     00009 ;               (3) Tested from 1200 to 19200 Baud(@ 16 & 20 Mhz)
                     00010 ;
                     00011 ;       The User gets a total time as specified by the User Cycles
                     00012 ;       in the table ( or from equations ). The user routine has to
                     00013 ;       exactly use up this amount of time. After this time the User
                     00014 ;       routine has to give up the control to the Operating System.
                     00015 ;       If less than 52 uS is used, then the user should wait in a
                     00016 ;       delay loop, until exactly 52 uS.
                     00017 ;
                     00018 ;       Transmission :
                     00019 ;               Transmit Data is output on DX pin (Bit DX of PORTA).
                     00020 ;               In the user routine, the user should load the
                     00021 ;               data to be transmitted in the XmtReg and Set the
                     00022 ;               X_flag ( bsf FlagRX,X_flag ). This flag gets cleared
                     00023 ;               after the transmission.
                     00024 ;
                     00025 ;       Reception :
                     00026 ;               Data is received on pin DR ( bit DR of PORTA ).
                     00027 ;               The User should constantly check the "R_done" flag
                     00028 ;               to see if reception is over. If the reception is
                     00029 ;               in progress, R_flag is set to 1.
                     00030 ;               If the reception is over, "R_done" flag is set to 1.
                     00031 ;               The "R_done" flag gets reset to zero when a next start
                     00032 ;               bit is detected. So, the user should constantly check
                     00033 ;               the R_done flag, and if SET, then the received word
                     00034 ;               is in Register "RcvReg". This register gets cleared
                     00035 ;               when a new start bit is detected.
                     00036 ;
                     00037 ;    Program Memory :
                     00038 ;       Total Program Memory Locations Used ( except initialization
                     00039 ;       in "main" & User routine ) = 132 locations.
                     00040 ;
                     00041 ;    Data Memory :
                     00042 ;       Total Data memory locations (file registers used) = 6
                     00043 ;               2 File registers to hold Xmt Data & Rcv Data
                     00044 ;               1 File registers for Xmt/Rcv flag test bits
                     00045 ;               3 File registers for delay count & scratch pad
                     00046 ;
                     00047 ;    Stack :
                     00048 ;       Only one level of stack is used in the Operating System/RS232
                     00049 ;       routine. But this is freed as soon as the program returns to
                     00050 ;       the user routine.
                     00051 ;
                     00052 ;    Timer0 :     Not Used
                     00053 ;    WDT  :     Not Used
                     00054 ;
                     00055 ;       Program:          RS232.ASM
```


© 1997 Microchip Technology Inc.

```
                00056 ;       Revision Date:
                00057 ;                          1-16-97    Compatibility with MPASMWIN 1.40
                00058 ;
                00059 ;*******************************************************************
                00060 ;
                00061         INCLUDE         <P16C5X.INC>
                00001         LIST
                00002 ; P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology, Inc.
                00224         LIST
                00062
000001FF        00063 PIC54   equ     1FFH    ; Define Reset Vector
00000001        00064 Same    equ     1
00000007        00065 MSB     equ     7
                00066
                00067         INCLUDE         <RS232.H>
                00001 ;*****************************************************************
                00002 ;                       RS232 Communication Parameters
                00003 ;
                00004 ;
00000001        00005 X_MODE  equ 1  ; If (X_MODE==1) Then transmit LSB first
                00006                ; if (X_MODE==0) Then transmit MSB first (CODEC like)
00000001        00007 R_MODE  equ 1  ; If (R_MODE==1) Then receive LSB first
                00008                ; if ( R_MODE==0) Then receive MSB first (CODEC like)
00000001        00009 X_Nbit  equ 1  ; if (X_Nbit==1)#of data bits (Transmission) is 8 else 7
00000001        00010 R_Nbit  equ 1  ; if (R_Nbit==1) # of data bits (Reception) is 8 else 7
                00011 ;
00000000        00012 SB2     equ 0  ; if SB2 = 0 then 1 Stop Bit
                00013 ;              ; if SB2 = 1 then 2 Stop Bit
                00014 ;*****************************************************************
                00015 ;       Transmit & Receive Test Bit Assignments
                00016 ;
00000000        00017 X_flag  equ 0  ; Bit 0 of FlagRX
00000002        00018 R_flag  equ 2  ; Bit 1 of FlagRX
00000003        00019 S_flag  equ 3  ; Bit 2 of FlagRX
00000004        00020 BitXsb  equ 4  ; Bit 3 of FlagRX
00000005        00021 A_flag  equ 5
00000006        00022 S_bit   equ 6  ; Xmt Stop Bit Flag( for 2/1 Stop bits )
                00023 ;
00000001        00024 R_done  equ 1  ; When Reception complete, this bit is SET
00000000        00025 X_done  equ X_flag  ; When Xmission complete, this bit is Cleared
                00026 ;
00000000        00027 DX      equ 0  ; Transmit Pin ( Bit 0 of Port A )
00000001        00028 DR      equ 1  ; Receive Pin ( Bit 1 of Port A )
                00029 ;
                00030 ;*********************  Data RAM Assignments  *********************
                00031 ;
0008            00032         ORG     08H     ; Dummy Origin
                00033 ;
0008            00034 RcvReg  RES     1       ; Data received
0009            00035 XmtReg  RES     1       ; Data to be transmitted
000A            00036 Xcount  RES     1       ; Counter for #of Bits Transmitted
000B            00037 Rcount  RES     1       ; Counter for #of Bits to be Received
000C            00038 DlyCnt  RES     1       ; Counter for Delay constant
000D            00039 FlagRX  RES     1       ; Transmit & Receive test flag hold register
                00040 ;_____________________________________________________________
                00041 ;    Constants       19200   9600    4800    2400    1200
                00042 ;  ( @ 20 Mhz )
                00043 ;_____________________________________________________________
                00044 ;       K0            0       13      57      143     317*
                00045 ;       K1            49      98      184     358*    705*
                00046 ;       K2            34      60      103     191     364*
                00047 ;       K3            27      53      96      184     357*
                00048 ;       K4            29      55      98      186     359*
                00049 ;       K5            30      56      99      187     360*
                00050 ;       K6            0       0       0       0       0
                00051 ;       K7            56      104     190     365*    712*
```

```
00052 ;
00053 ;    User Cycles    118     260      521     1042     2083
00054 ; ************************************************************
00055 ;
00056 ;
00057 ;_____________________________________________________________
00058 ;    Constants       19200   9600     4800     2400     1200
00059 ;  ( @ 8 Mhz )
00060 ;_____________________________________________________________
00061 ;        K0            --      0        5       39      109
00062 ;        K1            --     39       80      150      288*
00063 ;        K2            --     27       51       86      155
00064 ;        K3            --     21       44       80      148
00065 ;        K4            --     23       46       82      150
00066 ;        K5            --     24       47       83      151
00067 ;        K6            --      0        0        0        0
00068 ;        K7            --     45       86      156      295*
00069 ;
00070 ;    User_Cycles       --     86      208      416      832
00071 ; ************************************************************
00072 ;
00073 ;_____________________________________________________________
00074 ;    Constants       19200   9600     4800     2400     1200
00075 ;  ( @ 4 Mhz )
00076 ;_____________________________________________________________
00077 ;        K0            --      --       0        5       39
00078 ;        K1            --      --      39       80      150
00079 ;        K2            --      --      27       51       86
00080 ;        K3            --      --      21       44       80
00081 ;        K4            --      --      23       46       82
00082 ;        K5            --      --      24       47       83
00083 ;        K6            --      --       0        0        0
00084 ;        K7            --      --      45       86      156
00085 ;
00086 ;    User_Cycles       --      --      86      208      416
00087 ; ************************************************************
00088
00089 ;
00090 ; The constants marked " * " are >255. To implement these constants
00091 ; in delay loops, the delay loop should be broken into 2 or more loops.
00092 ; For example, 357 = 255+102. So 2 delay loops, one with 255 and
00093 ; the other with 102 may be used.
00094 ;*****************************************************************
00095 ;       Set Delay Constants for 9600 Baud @ CLKIN = 8 Mhz
00096 ;
00000000  00097 K0      EQU     .0
00000027  00098 K1      EQU     .39
0000001B  00099 K2      EQU     .27
00000015  00100 K3      EQU     .21
00000017  00101 K4      EQU     .23
00000018  00102 K5      EQU     .24
00000000  00103 K6      EQU     .0
0000002D  00104 K7      EQU     .45
00105 ;
00106 ;*****************************************************************
00107
00068 ;
0000      00069         ORG     0
00070 ;*****************************************************************
00071 ;
0000 0C01 00072 Delay   movlw   K0+1
0001 002C 00073         movwf   DlyCnt          ; Total Delay = 3K+6
0002 02EC 00074 redo    decfsz  DlyCnt,Same
0003 0A02 00075         goto    redo
0004 0800 00076         retlw   0
00077 ;
```


© 1997 Microchip Technology Inc.

```
0005 002C      00078 Delay1  movwf   DlyCnt
0006 02EC      00079 redo_1  decfsz  DlyCnt,Same   ;
0007 0A06      00080         goto    redo_1
0008 0A8D      00081         goto    User
               00082 ;
0009 002C      00083 Delay2  movwf   DlyCnt
000A 02EC      00084 redo_2  decfsz  DlyCnt,Same    ; Delay =   = 260 Cycles
000B 0A0A      00085         goto    redo_2
000C 0A67      00086         goto    User_1
               00087 ;
000D 0625      00088 R_strt  btfsc   PORTA,DR       ; check for a Start Bit
000E 0A17      00089         goto    ShellY         ; delay for 104/2 uS
000F 042D      00090         bcf     FlagRX,R_done  ; Reset Receive done flag
0010 054D      00091         bsf     FlagRX,R_flag  ; Set flag for Reception in Progress
0011 078D      00092         btfss   FlagRX,BitXsb
0012 05AD      00093         bsf     FlagRX,A_flag  ; A_flag is for start bit detected in R_strt
0013 0068      00094         clrf    RcvReg         ; Clear all bits of RcvReg
               00095         IF      R_Nbit
0014 0C08      00096         movlw   8              ; 8 Data bits
               00097         ELSE
               00098         movlw   7              ; 7 data bits
               00099         ENDIF
0015 002B      00100         movwf   Rcount
0016 0A78      00101         goto    Shell          ; delay for 104+104/4
               00102 ;
0017 078D      00103 ShellY  btfss   FlagRX,BitXsb
0018 0A78      00104         goto    Shell
0019 054D      00105         bsf     FlagRX,R_flag
001A 0A78      00106         goto    Shell
               00107 ;
001B 0403      00108 R_next  bcf     STATUS,C
               00109         IF      R_MODE
001C 0328      00110         rrf     RcvReg,Same    ; to set if MSB first or LSB first
               00111         ELSE
               00112         rlf     RcvReg,Same
               00113         ENDIF
001D 0625      00114         btfsc   PORTA,DR
               00115         IF      R_MODE
               00116           IF      R_Nbit
001E 05E8      00117           bsf     RcvReg,MSB   ; Conditional Assembly
               00118           ELSE
               00119           bsf     RcvReg,MSB-1
               00120           ENDIF
               00121         ELSE
               00122         bsf     RcvReg,LSB
               00123         ENDIF
001F 02EB      00124         decfsz  Rcount,Same
0020 0A78      00125         goto    Shell
0021 044D      00126         bcf     FlagRX,R_flag
0022 056D      00127         bsf     FlagRX,S_flag
0023 052D      00128         bsf     FlagRX,R_done
0024 0A78      00129         goto    Shell
               00130 ;
               00131 ;       Reception Done
               00132 ;
0025 0405      00133 X_strt  bcf     PORTA,DX       ; Send Start Bit
               00134         IF      X_Nbit
0026 0C08      00135         movlw   8
               00136         ELSE
               00137         movlw   7
               00138         ENDIF
0027 002A      00139         movwf   Xcount
               00140         IF      X_MODE
               00141         ELSE
               00142           IF    X_Nbit
               00143           ELSE
```


© 1997 Microchip Technology Inc.

```
                  00144           rlf   XmtReg,Same
                  00145           ENDIF
                  00146         ENDIF
0028 0A50         00147         goto    X_SB
                  00148 ;
0029 0403         00149 X_next  bcf     STATUS,C
                  00150         IF      X_MODE
002A 0329         00151         rrf     XmtReg,Same     ; Conditional Assembly
                  00152         ELSE                    ; to set if MSB first or LSB first
                  00153         rlf     XmtReg,Same
                  00154         ENDIF
002B 0603         00155         btfsc   STATUS,C
002C 0505         00156         bsf     PORTA,DX
002D 0703         00157         btfss   STATUS,C
002E 0405         00158         bcf     PORTA,DX
002F 00EA         00159         decf    Xcount,Same
0030 0A52         00160         goto    X_Data
                  00161 ;
0031 040D         00162 X_SB_1  bcf     FlagRX,X_flag   ; Xmt flag = 0 -- transmission over
0032 0C09         00163         movlw   9
0033 002A         00164         movwf   Xcount
0034 0505         00165         bsf     PORTA,DX        ; Send Stop Bit
0035 0A60         00166         goto    X_Stop
                  00167 ;
0036 0505         00168 X_SB_2  bsf     PORTA,DX
0037 04CD         00169         bcf     FlagRX,S_bit
0038 0A60         00170         goto    X_Stop
                  00171 ;
                  00172 ;   End of Transmission
                  00173 ;
0039 076D         00174 R0_X0   btfss   FlagRX,S_flag
003A 0A8D         00175         goto    User
003B 046D         00176         bcf     FlagRX,S_flag
003C 0900         00177         call    Delay
003D 0C2E         00178         movlw   K7+1
003E 0A05         00179         goto    Delay1
                  00180 ;
003F              00181 R1_X0
003F 0900         00182         call    Delay
0040 0C28         00183         movlw   K1+1            ; delay for 1st bit is 104+104/4
0041 002C         00184         movwf   DlyCnt
                  00185         IF      R_Nbit
0042 0C08         00186         movlw   8               ; 8 Data bits
                  00187         ELSE
                  00188         movlw   7               ; 7 data bits
                  00189         ENDIF
0043 018B         00190         xorwf   Rcount,W
0044 0643         00191         btfsc   STATUS,Z
0045 0A06         00192         goto    redo_1
0046 0C1C         00193         movlw   K2+1
0047 0A05         00194         goto    Delay1
                  00195 ;
0048              00196 R1_X1                           ; same as R0_X1
0048 0C09         00197 R0_X1   movlw   9
0049 008A         00198         subwf   Xcount,W
004A 0643         00199         btfsc   STATUS,Z
004B 0A25         00200         goto    X_strt
004C 022A         00201         movf    Xcount,Same     ; to check if All data bits Xmted
004D 0743         00202         btfss   STATUS,Z
004E 0A29         00203         goto    X_next
                  00204         IF      SB2
                  00205           btfsc FlagRX,S_bit
                  00206           goto  X_SB_2
                  00207           bsf   FlagRX,S_bit
                  00208           goto  X_SB_1
                  00209         ELSE
```


© 1997 Microchip Technology Inc.

```
004F 0A31      00210           goto    X_SB_1
               00211         ENDIF
               00212 ;
               00213 ;
0050 0A51      00214 X_SB    goto    cycle4
0051 0A52      00215 cycle4  goto    X_Data
               00216 ;
0052 06AD      00217 X_Data  btfsc   FlagRX,A_flag
0053 0A59      00218         goto    SbDly
0054 068D      00219         btfsc   FlagRX,BitXsb
0055 0A5D      00220         goto    ABC
0056 0900      00221         call    Delay
0057 0C16      00222         movlw   K3+1
0058 0A09      00223         goto    Delay2
               00224 ;
0059 04AD      00225 SbDly   bcf     FlagRX,A_flag
005A 0900      00226         call    Delay
005B 0C18      00227         movlw   K4+1
005C 0A09      00228         goto    Delay2
               00229 ;
005D 048D      00230 ABC     bcf     FlagRX,BitXsb
005E 0900      00231         call    Delay
005F 0A67      00232         goto    User_1
               00233 ;
0060           00234 X_Stop
0060 06AD      00235         btfsc   FlagRX,A_flag
0061 0A59      00236         goto    SbDly
0062 068D      00237         btfsc   FlagRX,BitXsb
0063 0A5D      00238         goto    ABC
0064 0900      00239         call    Delay
0065 0C19      00240         movlw   K5+1
0066 0A09      00241         goto    Delay2
               00242 ;
0067 064D      00243 User_1  btfsc   FlagRX,R_flag
0068 0A77      00244         goto    Sync_1          ; Reception already in progress
0069 066D      00245         btfsc   FlagRX,S_flag
006A 0A74      00246         goto    Sync_3
006B 0625      00247         btfsc   PORTA,DR       ; check for a Start Bit
006C 0A77      00248         goto    Sync_2          ; No Start Bit - goto User routine
006D 042D      00249         bcf     FlagRX,R_done   ; Reset Receive done flag
006E 044D      00250         bcf     FlagRX,R_flag
006F 058D      00251         bsf     FlagRX,BitXsb   ; Set flag for Reception in Progress
0070 0068      00252         clrf    RcvReg          ; Clear all bits of RcvReg
               00253         IF      R_Nbit
0071 0C08      00254         movlw   8               ; 8 Data bits
               00255         ELSE
               00256         movlw   7               ; 7 data bits
               00257         ENDIF
0072 002B      00258         movwf   Rcount
0073 0A8D      00259         goto    User
               00260 ;
0074 046D      00261 Sync_3  bcf     FlagRX,S_flag
0075 0C01      00262         movlw   K6+1
0076 0A05      00263         goto    Delay1
               00264 ;
0077           00265 Sync_1
0077 0A8D      00266 Sync_2  goto    User
               00267 ;
               00268 ;*****************************************************************
               00269 ;
0078 064D      00270 Shell   btfsc   FlagRX,R_flag
0079 0A7D      00271         goto    Chek_X
007A 060D      00272         btfsc   FlagRX,X_flag
007B 0A48      00273         goto    R0_X1
007C 0A39      00274         goto    R0_X0           ; Case for R0_X0
007D 060D      00275 Chek_X  btfsc   FlagRX,X_flag
```


© 1997 Microchip Technology Inc.

```
007E 0A48      00276         goto    R1_X1
007F 0A3F      00277         goto    R1_X0
               00278 ;
               00279 ;
               00280 ;*****************************************************************
               00281 ;       Operating System
               00282 ;  The User routine after time = B/2, should branch Here
               00283 ;
0080 074D      00284 Op_Sys  btfss   FlagRX,R_flag
0081 0A0D      00285         goto    R_strt
0082 0A1B      00286         goto    R_next
               00287 ;
               00288 ;*****************************************************************
               00289 ;
0083 0C0E      00290 main    movlw   0EH             ; Bit 0 of Port A is Output
0084 0005      00291         tris    PORTA           ; Set PORTA.0 as output ( DX )
               00292 ;                                 & PORTA.1 is input ( DR )
0085 0505      00293         bsf     PORTA,DX
0086 0C09      00294         movlw   9
0087 002A      00295         movwf   Xcount          ; If Xcount == 9, Then send start bit
0088 006D      00296         clrf    FlagRX          ; Clear All flag bits.
               00297         IF      SB2
               00298           bsf     FlagRX,S_bit  ; Set Xmt Stop bit flag(2 Stop Bits)
               00299         ELSE
0089 04CD      00300           bcf     FlagRX,S_bit  ; Clear Xmt Stop bit flag
               00301         ENDIF
008A 0C1F      00302         movlw   1FH             ; Prescaler = 4
008B 0002      00303         OPTION                  ; Set TMR0 increment on internal Clock
008C 0A80      00304         goto    Op_Sys
               00305 ;
               00306 ;*****************************************************************
               00307 ;
               00308 ;       ******************  User Routine  ***************
               00309 ;  The User routine should use up time exactly = User time as given
               00310 ;  in the Constants Table ( or by Equations for constants ).
               00311 ;  At 9600, this 86 Clock Cycles. TMR0 timer is used here to count
               00312 ;  upto 86 cycles ( From 128-86 To 0 ) by examining Bit 7 of TMR0.
               00313 ;
  00000030     00314 K_user  equ     .128+.6-.86
               00315 ;
008D 0C30      00316 User    movlw   K_user
008E 0021      00317         movwf   TMR0
008F 062D      00318         btfsc   FlagRX,R_done
0090 0A97      00319         goto    ErrChk
0091 060D      00320 SetXmt  btfsc   FlagRX,X_flag
0092 0A9C      00321         goto    Op
0093 0C41      00322         movlw   41H
0094 0029      00323         movwf   XmtReg
0095 050D      00324         bsf     FlagRX,X_flag   ; Enable Xmission
0096 0A9C      00325         goto    Op
               00326 ;
0097           00327 ErrChk
0097 0C5A      00328         movlw   "Z"
0098 0188      00329         xorwf   RcvReg,W
0099 0643      00330         btfsc   STATUS,Z
009A 0A91      00331         goto    SetXmt
009B 0A9B      00332 err1    goto    err1            ; Received word is not "Z"
               00333 ;
009C 07E1      00334 Op      btfss   TMR0,MSB        ; Test for TMR0 bit 7
009D 0A9C      00335         goto    Op              ; If Set, Then TMR0 has incremented
009E 0A80      00336 Oflow   goto    Op_Sys          ; to 128.
               00337 ;
               00338 ; *********************************************************
               00339 ;
01FF           00340         ORG     PIC54
01FF 0A83      00341         goto    main
```


© 1997 Microchip Technology Inc.

```
                00342
                00343           END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   160
Program Memory Words Free:   352

Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

PLD Replacement

Author: *Sumit Mitra*
Microchip Technology, Inc.

INTRODUCTION

PIC16C5X microcontrollers are ideal for implementing low cost combinational and sequential logic circuits that traditionally have been implemented using either numerous TTL gates or using programmable logic chips such as PLAs or EPLDs.

The PIC16C5X is a family of high-performance 8-bit microcontrollers from Microchip Technology Inc. It employs a Harvard architecture, (i.e., has a separate data bus [8-bit] and program bus [12-bit]). All instructions are single word and execute in one cycle except for program branches. The instruction cycle time is 200 ns at 20 MHz. PIC16C5X microcontrollers are ideal for PLD-type applications because:

- Very low cost
- Few external components required
- Fully programmable. PIC16C5X Microcontrollers are offered as One Time Programmable (OTP) EPROM devices.
- Available off the shelf from distributors
- Calibration in software for improved measurement accuracy
- Power savings using PIC16C5X's Sleep mode.
- PIC16C5X's output pins have large, current source/sink capability to drive LED's directly.

IMPLEMENTING A PLA

To implement a generic combinational logic function, we can simply emulate an AND-OR PLA in software. This requires the logic outputs to be described as sums of products. To describe our algorithm, we will use a simple 8-input, 8-output PLA with 24 product terms (Figure 1). We will further use the truth table in Figure 2 as the PLA function being implemented. In this example, only four inputs (A3:A0) are used and the other four inputs (A7:A4) are "don't care". On the output side, seven output pins (Y6:Y0) are used and Y7 is unused. To implement this PLA, the logic inputs A0,A1,...,A7 can be connected to PORTB pins RB0,RB1,...,RB7 respectively. The logic outputs Y0,Y1,...,Y7 will appear on PORTC pins RC0,RC1,...,RC7 respectively. PORTB is configured as input and PORTC will be configured as output. To evaluate each product term one XOR (exclusive .OR.) and one .AND. operation will be required. For example, to determine product term P3 = A3.A2.A1.A0, the expression to evaluate is:

(A7:A0 .XOR. xxxx0011b) .AND. 00001111b).

The constant with which .XOR. is performed will be referred to as P3_x in our discussion. P3_x = xxxx0011b will ensure that if A3:A0 = 0011b, then the least significant 4-bits of the result will be 0000b. The .AND. constant, referred to here as P3_a (Product term 3, .AND. constant) basically eliminates the "don't care" inputs (here A7:A4) by masking them. Therefore, if the result of the .XOR.-.AND. operation is zero then P3 = 1 else P3 = 0. Once the Product terms are evaluated they are stored in four product registers Preg_0 to Preg_3. To determine an output term:

$$Y0 = P0 + P2 + P3 + P5 + P6 + P7 + P8 + P9 + P10 + P12 + P13 + P14$$

we need to evaluate the following expression:

(Preg_a .AND. OR_a0) .OR. (Preg_b .AND. OR_b0) .OR. (Preg_c .AND. OR_c0)

In our case the constant values to implement Y0 are as follows:

OR_a0 =	1	1	1	0	1	1	0	1
	P7	P6	P5		P3	P2		P0
OR_b0 =	1	1	0	1	0	1	1	1
OR_c0 =	0	0	0	0	0	0	0	0

For a larger number of inputs, outputs, or product terms, the evaluation will be more complex but follows the same principle. Appendix A shows the assembly code to implement this 8 input x 8 output x 24 Product PLA. This example optimizes speed as well as program memory requirements. Appendix B shows a slightly different implementation (only the EVAL_Y MACRO is different) that optimizes program memory usage over speed. Table 1 shows the time and resources required to implement different size PLAs.

© 1997 Microchip Technology Inc.

FIGURE 1: A SIMPLE PLA

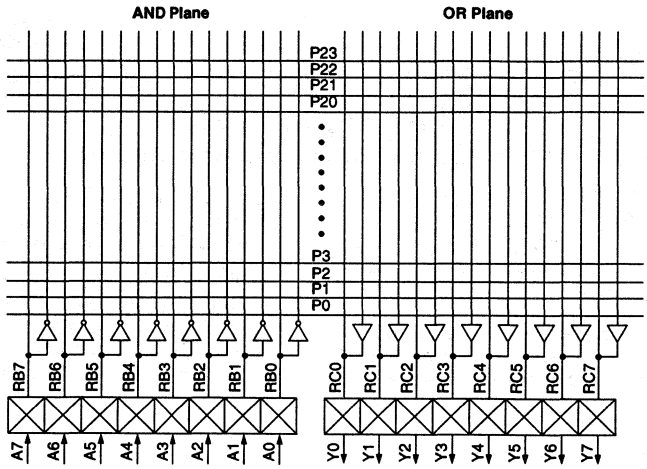
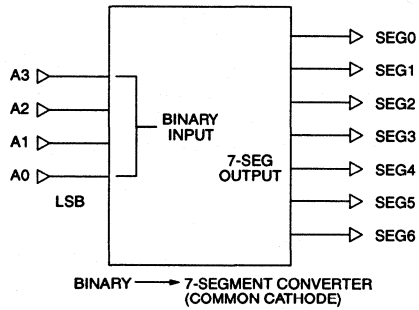
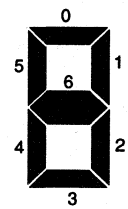


FIGURE 2: BINARY TO 7-SEGMENT CONVERSION EXAMPLE





Truth Table:

HEX	A3	A2	A1	A0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Product Terms
0	0	0	0	0	1	1	1	1	1	1	0	P0 = A3, A2, A1, A0
1	0	0	0	1	0	1	1	0	0	0	0	P1
2	0	0	1	0	1	1	0	1	1	0	1	P2
3	0	1	1	1	1	1	1	1	0	0	1	P3
4	0	1	0	0	0	1	1	0	0	1	1	P4
5	0	1	0	1	1	0	1	1	0	1	1	P5
6	0	1	1	0	1	0	1	1	1	1	1	P6
7	0	1	1	1	1	1	1	0	0	0	0	P7
8	1	0	0	0	1	1	1	1	1	1	1	P8
9	1	0	0	1	1	1	1	0	0	1	1	P9
A	1	0	1	0	1	1	1	0	1	1	1	P10
B	1	0	1	1	0	0	0	1	1	1	1	P11
C	1	1	0	0	1	0	0	1	1	1	0	P12
D	1	1	0	1	0	1	1	1	1	0	1	P13
E	1	1	1	0	1	0	0	1	1	1	1	P14
F	1	1	1	1	1	0	0	0	1	1	1	P15 = A3, A2, A1, A0

© 1997 Microchip Technology Inc.

SPEED/RESPONSE TIME

The worst case response time of a PLA implemented in this fashion can be calculated as follows. First, we define TD = time required to execute the PLA program assuming the worst case program branches are taken. Then the maximum propagation delay time from input change to valid output = 2TD. This is because if an input changes just after the program reads the input port, its effect will not show up until the program completes the current execution cycle, re-reads the input and recalculates the output. This is shown in Figure 3. There are ways to improve the delay time, such as sample inputs several times throughout the PLA program and if an input change is sensed, return to the beginning rather than execute the rest of the evaluation code.

A Table Lookup Method for Small PLA Implementation

If the number of inputs is small (8 or less) then a simple table lookup method can be used to implement the PLA. This will improve execution time to around 5 µs (@ 8 MHz input clock). The following code implements the Binary Coded Decimal (BCD) to 7-segment conversion (Figure 2) using this technique.

EXAMPLE 1: LOOKUP TABLE – PLA METHOHD

```
begin   movlw   0ffh        ;
        tris    6           ;Port_b = input
        clrw                ;
        tris    7           ;Port_c = output
pla 88  movf    Port_b,w    ;Read input
        andlw   0fh         ;Mask off bits 7:4
        call    op_tbl      ;
        movwf   Port_c      ;Write output
        goto    pla88       ;
op_tbl  addwf   pc          ;Computed jump for
                            ;table lookup
        retlw   b"00111111";
        retlw   b"00000110";
        retlw   b"01011011";
        retlw   b"01001111";
        retlw   b"01100110";
        retlw   b"01101101";
        retlw   b"01111101";
        retlw   b"00000111";
        retlw   b"01111111";
        retlw   b"01100111";
        retlw   b"01110111";
        retlw   b"01111000";
        retlw   b"00111001";
        retlw   b"01011110";
        retlw   b"01111001";
        retlw   b"01110001";
```

TABLE 1: EXECUTION TIME AND RESOURCES NECESSARY FOR DIFFERENT SIZE PLAs

Number of Inputs Including FDBK	Number of Outputs Including FDBK and O/E Control	Number of Products	Number of RAM Locations Required NRAM	Number of Memory Locations Required NROM	Number of Instruction Cycles to Execute PLA NCYC	Real-time @ 20 MHz to Execute PLA	
8	8	24	5	228	228	45.6 µs	Time Efficient
			10	222	352	70.4 µs	Code Efficient
8	8	48	8	447	447	89.8 µs	Time Efficient
			13	384	535	107 µs	Code Efficient
16	16	64	12	1042	1042	208.4 µs	Time Efficient
			22	843	1250	250 µs	Code Efficient
20	24	80	16	1661	1661	372.2 µs	Time Efficient
			40	1462	2221	444.2 µs	Code Efficient

If NI = Number of inputs

NIW = Number of input words, $N_{IW} = \frac{N_I}{8}$

NP = Number of products

NPW = Number of product words, i.e. $N_{PW} = \frac{N_P}{8}$

NO = Number of output words

NOW = Number of output words, i.e., $N_{OW} = \frac{N_O}{8}$

Then, NRAM @ NIW + NPW : Time Efficient

NRAM @ NIW + NOW + NPW + 2: Code Efficient

NROM @ 8 + NPW + NOW + NP [2 + 3NIW] + NO • 4: Time Efficient

NROM @ 17 + NPW + NOW + NP [2 + 3NIW] + NO [NPW + 3]: Code Efficient

NCYC @ 8 + NPW + NOW + NP[2 + 3NIW] + NO [2NPW • 4]: Time Efficient

NCYC @ 8 + NPW + NOW + NP [2 + 3NIW] + 5N [NPW + 1]: Code Efficient

FIGURE 3: PLA PROGRAM FLOW

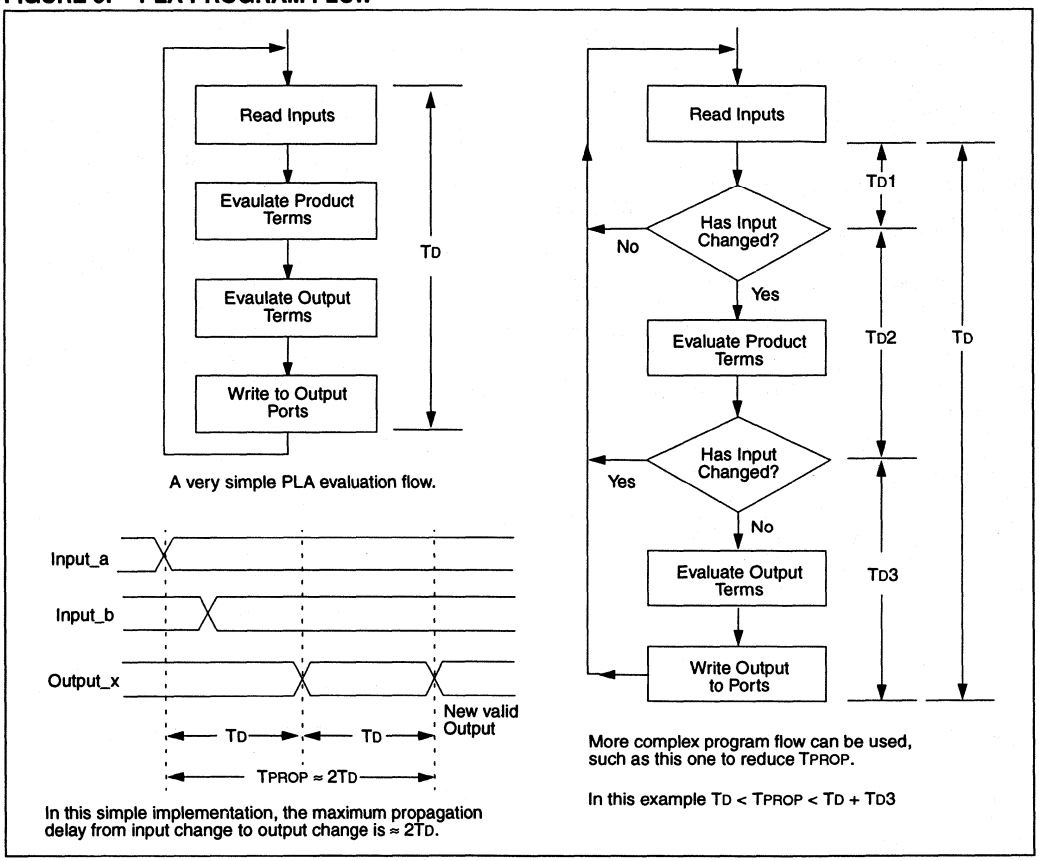
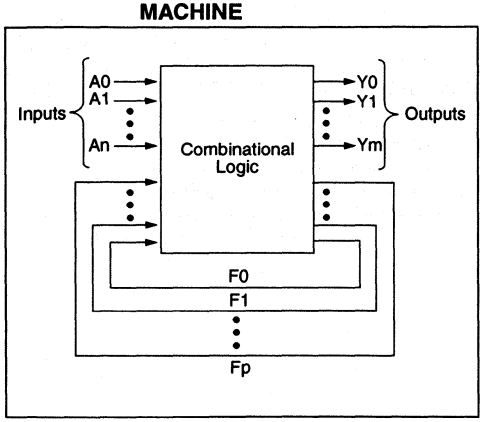


FIGURE 4: ASYNCHRONOUS STATE MACHINE

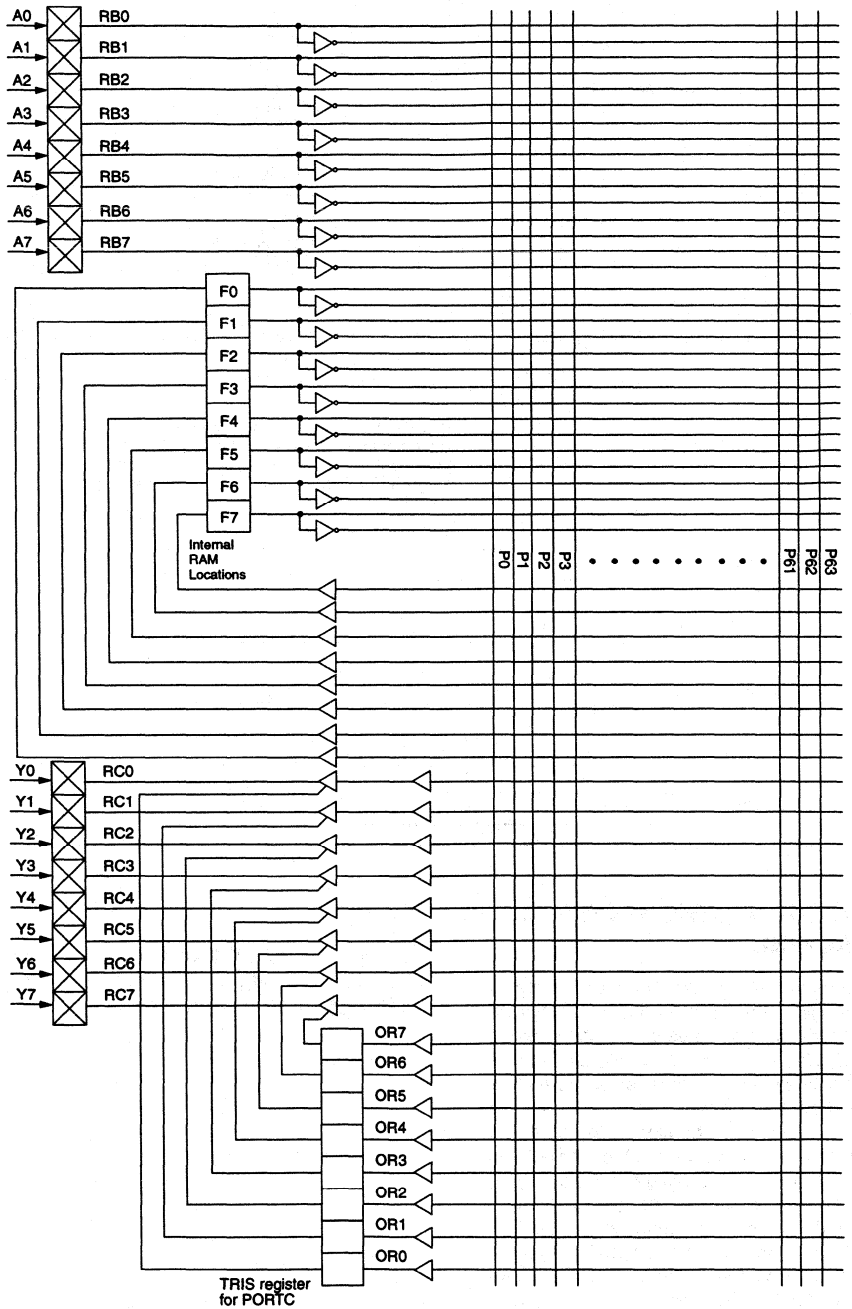


IMPLEMENTING AN ASYNCHRONOUS STATE MACHINE

The concept can be easily extended to implement sequential logic (i.e., a state machine). Figure 4 shows a state machine with n inputs (A0:An), m outputs (Y0:Ym) and p states that feedback as inputs to the PLA (F0:Fp). In a PIC16C5X the states will be stored as bits in RAM location. Input will now mean input from a port as well as from the feedback registers. Figure 5 shows an example PLA with eight inputs A0,A1,..,A7 that are connected to PORTB pins RB0,RB1,..,RB7. This example PLA has a total of 24 outputs of which eight are actual outputs, another eight are output enable control for the outputs and the other eight are feedbacks (or states). The PLA shown here, therefore, in essence implements an asynchronous state machine (i.e., there is no system clock).

© 1997 Microchip Technology Inc.

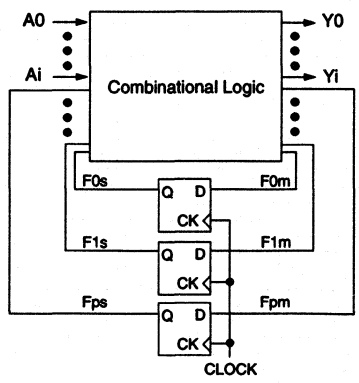
FIGURE 5: EXAMPLE OF A LARGER PLA IMPLEMENTATION

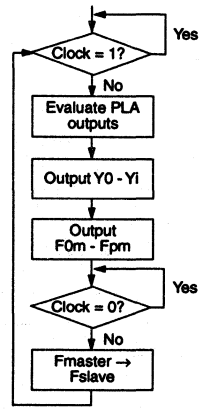


This example shows 16 inputs (including feedback), 64 product terms and 24 outputs including feedback and O/E control. This example demonstrates that O/E control is easily implementable using PIC16C5X's bi-directional ports with Tri-State® control.

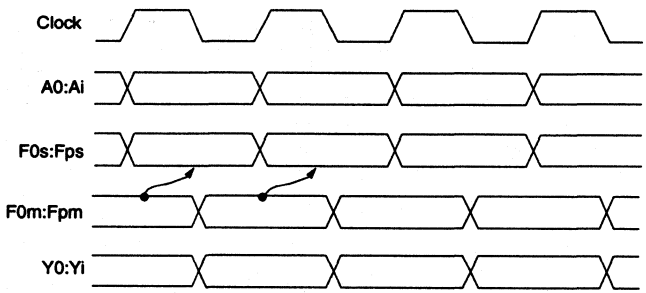
Tri-State® is a registered trademark of National Semiconductor Corporation

FIGURE 6: SYNCHRONOUS STATE MACHINE IMPLEMENTATION





Flowchart for implementing synchronous state machine. Clock is input to an I/O port. Fmaster and Fslave are RAM locations that store state variables F0m, ... Fpm and F0s, ... Fps respectively.



IMPLEMENTING A SYNCHRONOUS STATE MACHINE

In a synchronous system (Figure 6) usually all inputs are stable at the falling (or rising) edge of the system clock. The state machine samples inputs on the falling edge, then evaluates state and output information. The state outputs are latched by the rising edge of the clock before feeding them back to the input (so that they are stable at the falling edge of the clock). To implement such a state machine, the system clock will have to be polled by an input pin. When a falling edge is detected, the PLA evaluation procedure will be invoked to compute outputs and write them to output pins. The PLA procedure will also determine the new state variables, F0m, F1m,...,Fpm and store them in RAM. The program will then wait until a rising edge on the clock input is detected and copy the "master" state variables (F0m,..,Fpm) to slave state variables (F0s,F1s,..,Fps). This step emulates the feedback flip-flops.

SUMMARY

In conclusion, the PIC16C5X can implement a generic PLA equation and provide a quick, low cost solution where system operation speed is not critical.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: PLA IMPLEMENTATION: TIME EFFICIENT APPROACH

```
MPASM 01.40 Released            PLA1A.ASM   1-16-1997  12:28:25         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ; pla1a.asm :
                     00005 ; This procedure implements a simple AND-OR PLA with:
                     00006 ;
                     00007 ;       8  inputs         := A7 A6 A5 A4 A3 A2 A1 A0
                     00008 ;       24 product terms  := P23 P22 ..... P0
                     00009 ;       8  outputs        := Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
                     00010 ;
                     00011 ; The eight inputs are assumed to be connected to PORT RB such that
                     00012 ; RB0 = A0, RB1 = A1, ... , RB7 = A7.
                     00013 ; The outputs are programmed to appear on port RC such that
                     00014 ; RC0 = Y0, RC1 = Y1, ... , RC7 = Y7.
                     00015 ;
                     00016 ; This implementation optimizes both speed & program memory usage
                     00017 ;
                     00018 ;       Program:          PLA1A.ASM
                     00019 ;       Revision Date:
                     00020 ;                         1-13-97   Compatibility with MPASMWIN 1.40
                     00021 ;
                     00022 ;*****************************************************************
                     00023 ;
                     00024 ; define RAM locations used:
                     00025 ;
0000000C             00026 input   equ  d'12'   ; RAM location 12 holds input
0000000D             00027 Y_reg   equ  d'13'   ; holds output result
                     00028
0000000E             00029 Preg_a  equ  d'14'   ; Product terms P0 to P7. Preg_a<0> = P0
0000000F             00030 Preg_b  equ  d'15'   ; Product terms P8 to P15. Preg_b<0> = P8
00000010             00031 Preg_c  equ  d'16'   ; Product terms P16 to P23. Preg_c<0> = P16
                     00032
                     00033 ; define some constants and file addresses:
                     00034 ;
00000000             00035 bit0    equ  0       ;
00000001             00036 bit1    equ  1       ;
00000002             00037 bit2    equ  2       ;
00000003             00038 bit3    equ  3       ;
00000004             00039 bit4    equ  4       ;
00000005             00040 bit5    equ  5       ;
00000006             00041 bit6    equ  6       ;
00000007             00042 bit7    equ  7       ;
                     00043 ;
00000003             00044 status  equ  3       ;
00000006             00045 port_b  equ  6       ;
00000007             00046 port_c  equ  7       ;
                     00047 ;
                     00048 ; define the AND plane programming variables:
                     00049 ;
00000000             00050 P0_x    equ  b'00000000'     ;
0000000F             00051 P0_a    equ  b'00001111'     ;
00000001             00052 P1_x    equ  b'00000001'     ;
0000000F             00053 P1_a    equ  b'00001111'     ;
00000002             00054 P2_x    equ  b'00000010'     ;
```

```
0000000F          00055 P2_a    equ  b'00001111'     ;
00000003          00056 P3_x    equ  b'00000011'     ;
0000000F          00057 P3_a    equ  b'00001111'     ;
00000004          00058 P4_x    equ  b'00000100'     ;
0000000F          00059 P4_a    equ  b'00001111'     ;
00000005          00060 P5_x    equ  b'00000101'     ;
0000000F          00061 P5_a    equ  b'00001111'     ;
00000006          00062 P6_x    equ  b'00000110'     ;
0000000F          00063 P6_a    equ  b'00001111'     ;
00000007          00064 P7_x    equ  b'00000111'     ;
0000000F          00065 P7_a    equ  b'00001111'     ;
00000008          00066 P8_x    equ  b'00001000'     ;
0000000F          00067 P8_a    equ  b'00001111'     ;
00000009          00068 P9_x    equ  b'00001001'     ;
0000000F          00069 P9_a    equ  b'00001111'     ;
0000000A          00070 P10_x   equ  b'00001010'     ;
0000000F          00071 P10_a   equ  b'00001111'     ;
0000000B          00072 P11_x   equ  b'00001011'     ;
0000000F          00073 P11_a   equ  b'00001111'     ;
0000000C          00074 P12_x   equ  b'00001100'     ;
0000000F          00075 P12_a   equ  b'00001111'     ;
0000000D          00076 P13_x   equ  b'00001101'     ;
0000000F          00077 P13_a   equ  b'00001111'     ;
0000000E          00078 P14_x   equ  b'00001110'     ;
0000000F          00079 P14_a   equ  b'00001111'     ;
0000000F          00080 P15_x   equ  b'00001111'     ;
0000000F          00081 P15_a   equ  b'00001111'     ;
00000000          00082 P16_x   equ  b'00000000'     ;
00000000          00083 P16_a   equ  b'00000000'     ;
00000000          00084 P17_x   equ  b'00000000'     ;
00000000          00085 P17_a   equ  b'00000000'     ;
00000000          00086 P18_x   equ  b'00000000'     ;
00000000          00087 P18_a   equ  b'00000000'     ;
00000000          00088 P19_x   equ  b'00000000'     ;
00000000          00089 P19_a   equ  b'00000000'     ;
00000000          00090 P20_x   equ  b'00000000'     ;
00000000          00091 P20_a   equ  b'00000000'     ;
00000000          00092 P21_x   equ  b'00000000'     ;
00000000          00093 P21_a   equ  b'00000000'     ;
00000000          00094 P22_x   equ  b'00000000'     ;
00000000          00095 P22_a   equ  b'00000000'     ;
00000000          00096 P23_x   equ  b'00000000'     ;
00000000          00097 P23_a   equ  b'00000000'     ;
                  00098
                  00099 ; define OR plane programming variables:
                  00100
000000ED          00101 OR_a0   equ  b'11101101'     ; for output Y0
000000D7          00102 OR_b0   equ  b'11010111'     ;
00000000          00103 OR_c0   equ  b'00000000'     ;
0000009F          00104 OR_a1   equ  b'10011111'     ; for output Y1
00000027          00105 OR_b1   equ  b'00100111'     ;
00000000          00106 OR_c1   equ  b'00000000'     ;
000000FB          00107 OR_a2   equ  b'11111011'     ; for output Y2
0000002F          00108 OR_b2   equ  b'00101111'     ;
00000000          00109 OR_c2   equ  b'00000000'     ;
0000006D          00110 OR_a3   equ  b'01101101'     ; for output Y3
00000079          00111 OR_b3   equ  b'01111001'     ;
00000000          00112 OR_c3   equ  b'00000000'     ;
00000045          00113 OR_a4   equ  b'01000101'     ; for output Y4
000000FD          00114 OR_b4   equ  b'11111101'     ;
00000000          00115 OR_c4   equ  b'00000000'     ;
00000071          00116 OR_a5   equ  b'01110001'     ; for output Y5
000000DF          00117 OR_b5   equ  b'11011111'     ;
00000000          00118 OR_c5   equ  b'00000000'     ;
0000007C          00119 OR_a6   equ  b'01111100'     ; for output Y6
000000EF          00120 OR_b6   equ  b'11101111'     ;
```


© 1997 Microchip Technology Inc.

```
  00000000          00121 OR_c6    equ  b'00000000'     ;
  00000000          00122 OR_a7    equ  b'00000000'     ; for output Y7
  00000000          00123 OR_b7    equ  b'00000000'     ;
  00000000          00124 OR_c7    equ  b'00000000'     ;
                    00125
                    00126
01FF                00127          org  01ffh                    ;
01FF 0A00           00128 begin    goto main                     ;
                    00129
0000                00130                 org     000h           ;
                    00131 ; define macro to evaluate 1 product (AND) term:
                    00132 ;
0000 0902           00133 main     call  pla88                   ;
0001 0A00           00134          goto  main                    ;
                    00135 ;
                    00136
                    00137 EVAL_P  MACRO   Preg_x,bit_n,Pn_x,Pn_a
                    00138                 movf    input,W         ;
                    00139                 xorlw   Pn_x            ;
                    00140                 andlw   Pn_a            ;
                    00141                 btfsc   status,bit2     ; skip if zero bit not set
                    00142                 bsf     Preg_x,bit_n    ; product term = 1
                    00143                 ENDM
                    00144
                    00145
                    00146 ; define macro to load OR term constants:
                    00147 ;
                    00148 EVAL_Y  MACRO   OR_an,OR_bn,OR_cn,bit_n
                    00149                 LOCAL   SETBIT  ;
                    00150                 movf    Preg_a,W        ;
                    00151                 andlw   OR_an   ;
                    00152                 btfss   status,bit2     ;
                    00153                 goto    SETBIT          ;
                    00154
                    00155                 movf    Preg_b,W        ;
                    00156                 andlw   OR_bn   ;
                    00157                 btfss   status,bit2     ;
                    00158                 goto    SETBIT  ;
                    00159
                    00160                 movf    Preg_c,W        ;
                    00161                 andlw   OR_cn   ;
                    00162                 btfss   status,bit2     ;
                    00163 SETBIT  bsf             Y_reg,bit_n     ;
                    00164                 ENDM
                    00165
                    00166 ; now the  PLA evaluation procedure:
                    00167 ;
0002 0CFF           00168 pla88   movlw   0ffh                    ;
0003 0006           00169                 tris    6               ; port_b = input
0004 0206           00170                 movf    port_b,W        ; read input
0005 002C           00171                 movwf   input           ; store input in a register
0006 006E           00172                 clrf    Preg_a          ; clear Product register a
0007 006F           00173                 clrf    Preg_b          ; clear Product register b
0008 0070           00174                 clrf    Preg_c          ; clear Product register c
0009 006D           00175                 clrf    Y_reg           ; clear output register
                    00176
                    00177 and_p1  EVAL_P  Preg_a,bit0,P0_x,P0_a
000A 020C               M                 movf    input,W         ;
000B 0F00               M                 xorlw   P0_x            ;
000C 0E0F               M                 andlw   P0_a            ;
000D 0643               M                 btfsc   status,bit2     ; skip if zero bit not set
000E 050E               M                 bsf     Preg_a,bit0     ; product term = 1
                    00178                 EVAL_P  Preg_a,bit1,P1_x,P1_a
000F 020C               M                 movf    input,W         ;
0010 0F01               M                 xorlw   P1_x            ;
0011 0E0F               M                 andlw   P1_a            ;
```

```
0012 0643          M          btfsc   status,bit2     ; skip if zero bit not set
0013 052E          M          bsf     Preg_a,bit1     ; product term = 1
                   00179      EVAL_P  Preg_a,bit2,P2_x,P2_a
0014 020C          M          movf    input,W         ;
0015 0F02          M          xorlw   P2_x            ;
0016 0E0F          M          andlw   P2_a            ;
0017 0643          M          btfsc   status,bit2     ; skip if zero bit not set
0018 054E          M          bsf     Preg_a,bit2     ; product term = 1
                   00180      EVAL_P  Preg_a,bit3,P3_x,P3_a
0019 020C          M          movf    input,W         ;
001A 0F03          M          xorlw   P3_x            ;
001B 0E0F          M          andlw   P3_a            ;
001C 0643          M          btfsc   status,bit2     ; skip if zero bit not set
001D 056E          M          bsf     Preg_a,bit3     ; product term = 1
                   00181      EVAL_P  Preg_a,bit4,P4_x,P4_a
001E 020C          M          movf    input,W         ;
001F 0F04          M          xorlw   P4_x            ;
0020 0E0F          M          andlw   P4_a            ;
0021 0643          M          btfsc   status,bit2     ; skip if zero bit not set
0022 058E          M          bsf     Preg_a,bit4     ; product term = 1
                   00182      EVAL_P  Preg_a,bit5,P5_x,P5_a
0023 020C          M          movf    input,W         ;
0024 0F05          M          xorlw   P5_x            ;
0025 0E0F          M          andlw   P5_a            ;
0026 0643          M          btfsc   status,bit2     ; skip if zero bit not set
0027 05AE          M          bsf     Preg_a,bit5     ; product term = 1
                   00183      EVAL_P  Preg_a,bit6,P6_x,P6_a
0028 020C          M          movf    input,W         ;
0029 0F06          M          xorlw   P6_x            ;
002A 0E0F          M          andlw   P6_a            ;
002B 0643          M          btfsc   status,bit2     ; skip if zero bit not set
002C 05CE          M          bsf     Preg_a,bit6     ; product term = 1
                   00184      EVAL_P  Preg_a,bit7,P7_x,P7_a
002D 020C          M          movf    input,W         ;
002E 0F07          M          xorlw   P7_x            ;
002F 0E0F          M          andlw   P7_a            ;
0030 0643          M          btfsc   status,bit2     ; skip if zero bit not set
0031 05EE          M          bsf     Preg_a,bit7     ; product term = 1
                   00185
                   00186      EVAL_P  Preg_b,bit0,P8_x,P8_a
0032 020C          M          movf    input,W         ;
0033 0F08          M          xorlw   P8_x            ;
0034 0E0F          M          andlw   P8_a            ;
0035 0643          M          btfsc   status,bit2     ; skip if zero bit not set
0036 050F          M          bsf     Preg_b,bit0     ; product term = 1
                   00187      EVAL_P  Preg_b,bit1,P9_x,P9_a
0037 020C          M          movf    input,W         ;
0038 0F09          M          xorlw   P9_x            ;
0039 0E0F          M          andlw   P9_a            ;
003A 0643          M          btfsc   status,bit2     ; skip if zero bit not set
003B 052F          M          bsf     Preg_b,bit1     ; product term = 1
                   00188      EVAL_P  Preg_b,bit2,P10_x,P10_a
003C 020C          M          movf    input,W         ;
003D 0F0A          M          xorlw   P10_x           ;
003E 0E0F          M          andlw   P10_a           ;
003F 0643          M          btfsc   status,bit2     ; skip if zero bit not set
0040 054F          M          bsf     Preg_b,bit2     ; product term = 1
                   00189      EVAL_P  Preg_b,bit3,P11_x,P11_a
0041 020C          M          movf    input,W         ;
0042 0F0B          M          xorlw   P11_x           ;
0043 0E0F          M          andlw   P11_a           ;
0044 0643          M          btfsc   status,bit2     ; skip if zero bit not set
0045 056F          M          bsf     Preg_b,bit3     ; product term = 1
                   00190      EVAL_P  Preg_b,bit4,P12_x,P12_a
0046 020C          M          movf    input,W         ;
0047 0F0C          M          xorlw   P12_x           ;
```


© 1997 Microchip Technology Inc.

```
0048 0E0F          M          andlw   P12_a            ;
0049 0643          M          btfsc   status,bit2      ; skip if zero bit not set
004A 058F          M          bsf     Preg_b,bit4      ; product term = 1
                   00191      EVAL_P  Preg_b,bit5,P13_x,P13_a
004B 020C          M          movf    input,W          ;
004C 0F0D          M          xorlw   P13_x            ;
004D 0E0F          M          andlw   P13_a            ;
004E 0643          M          btfsc   status,bit2      ; skip if zero bit not set
004F 05AF          M          bsf     Preg_b,bit5      ; product term = 1
                   00192      EVAL_P  Preg_b,bit6,P14_x,P14_a
0050 020C          M          movf    input,W          ;
0051 0F0E          M          xorlw   P14_x            ;
0052 0E0F          M          andlw   P14_a            ;
0053 0643          M          btfsc   status,bit2      ; skip if zero bit not set
0054 05CF          M          bsf     Preg_b,bit6      ; product term = 1
                   00193      EVAL_P  Preg_b,bit7,P15_x,P15_a
0055 020C          M          movf    input,W          ;
0056 0F0F          M          xorlw   P15_x            ;
0057 0E0F          M          andlw   P15_a            ;
0058 0643          M          btfsc   status,bit2      ; skip if zero bit not set
0059 05EF          M          bsf     Preg_b,bit7      ; product term = 1
                   00194
                   00195      EVAL_P  Preg_c,bit0,P16_x,P16_a
005A 020C          M          movf    input,W          ;
005B 0F00          M          xorlw   P16_x            ;
005C 0E00          M          andlw   P16_a            ;
005D 0643          M          btfsc   status,bit2      ; skip if zero bit not set
005E 0510          M          bsf     Preg_c,bit0      ; product term = 1
                   00196      EVAL_P  Preg_c,bit1,P17_x,P17_a
005F 020C          M          movf    input,W          ;
0060 0F00          M          xorlw   P17_x            ;
0061 0E00          M          andlw   P17_a            ;
0062 0643          M          btfsc   status,bit2      ; skip if zero bit not set
0063 0530          M          bsf     Preg_c,bit1      ; product term = 1
                   00197      EVAL_P  Preg_c,bit2,P18_x,P18_a
0064 020C          M          movf    input,W          ;
0065 0F00          M          xorlw   P18_x            ;
0066 0E00          M          andlw   P18_a            ;
0067 0643          M          btfsc   status,bit2      ; skip if zero bit not set
0068 0550          M          bsf     Preg_c,bit2      ; product term = 1
                   00198      EVAL_P  Preg_c,bit3,P19_x,P19_a
0069 020C          M          movf    input,W          ;
006A 0F00          M          xorlw   P19_x            ;
006B 0E00          M          andlw   P19_a            ;
006C 0643          M          btfsc   status,bit2      ; skip if zero bit not set
006D 0570          M          bsf     Preg_c,bit3      ; product term = 1
                   00199      EVAL_P  Preg_c,bit4,P20_x,P20_a
006E 020C          M          movf    input,W          ;
006F 0F00          M          xorlw   P20_x            ;
0070 0E00          M          andlw   P20_a            ;
0071 0643          M          btfsc   status,bit2      ; skip if zero bit not set
0072 0590          M          bsf     Preg_c,bit4      ; product term = 1
                   00200      EVAL_P  Preg_c,bit5,P21_x,P21_a
0073 020C          M          movf    input,W          ;
0074 0F00          M          xorlw   P21_x            ;
0075 0E00          M          andlw   P21_a            ;
0076 0643          M          btfsc   status,bit2      ; skip if zero bit not set
0077 05B0          M          bsf     Preg_c,bit5      ; product term = 1
                   00201      EVAL_P  Preg_c,bit6,P22_x,P22_a
0078 020C          M          movf    input,W          ;
0079 0F00          M          xorlw   P22_x            ;
007A 0E00          M          andlw   P22_a            ;
007B 0643          M          btfsc   status,bit2      ; skip if zero bit not set
007C 05D0          M          bsf     Preg_c,bit6      ; product term = 1
                   00202      EVAL_P  Preg_c,bit7,P23_x,P23_a
007D 020C          M          movf    input,W          ;
```


© 1997 Microchip Technology Inc.

```
007E 0F00                M                xorlw   P23_x           ;
007F 0E00                M                andlw   P23_a           ;
0080 0643                M                btfsc   status,bit2     ; skip if zero bit not set
0081 05F0                M                bsf     Preg_c,bit7     ; product term = 1
                       00203
                       00204 or_pl       EVAL_Y  OR_a0,OR_b0,OR_c0,bit0
  0000                   M                LOCAL   SETBIT          ;
0082 020E                M                movf    Preg_a,W        ;
0083 0EED                M                andlw   OR_a0           ;
0084 0743                M                btfss   status,bit2     ;
0085 0A8D                M                goto    SETBIT          ;
                         M
0086 020F                M                movf    Preg_b,W        ;
0087 0ED7                M                andlw   OR_b0   ;
0088 0743                M                btfss   status,bit2     ;
0089 0A8D                M                goto    SETBIT          ;
                         M
008A 0210                M                movf    Preg_c,W        ;
008B 0E00                M                andlw   OR_c0           ;
008C 0743                M                btfss   status,bit2     ;
008D 050D                M SETBIT         bsf     Y_reg,bit0      ;
                       00205              EVAL_Y  OR_a1,OR_b1,OR_c1,bit1
  0000                   M                LOCAL   SETBIT          ;
008E 020E                M                movf    Preg_a,W        ;
008F 0E9F                M                andlw   OR_a1           ;
0090 0743                M                btfss   status,bit2     ;
0091 0A99                M                goto    SETBIT          ;
                         M
0092 020F                M                movf    Preg_b,W        ;
0093 0E27                M                andlw   OR_b1           ;
0094 0743                M                btfss   status,bit2     ;
0095 0A99                M                goto    SETBIT          ;
                         M
0096 0210                M                movf    Preg_c,W        ;
0097 0E00                M                andlw   OR_c1           ;
0098 0743                M                btfss   status,bit2     ;
0099 052D                M SETBIT         bsf     Y_reg,bit1      ;
                       00206              EVAL_Y  OR_a2,OR_b2,OR_c2,bit2
  0000                   M                LOCAL   SETBIT          ;
009A 020E                M                movf    Preg_a,W        ;
009B 0EFB                M                andlw   OR_a2           ;
009C 0743                M                btfss   status,bit2     ;
009D 0AA5                M                goto    SETBIT          ;
                         M
009E 020F                M                movf    Preg_b,W        ;
009F 0E2F                M                andlw   OR_b2           ;
00A0 0743                M                btfss   status,bit2     ;
00A1 0AA5                M                goto    SETBIT  ;
                         M
00A2 0210                M                movf    Preg_c,W        ;
00A3 0E00                M                andlw   OR_c2           ;
00A4 0743                M                btfss   status,bit2     ;
00A5 054D                M SETBIT         bsf     Y_reg,bit2      ;
                       00207              EVAL_Y  OR_a3,OR_b3,OR_c3,bit3
  0000                   M                LOCAL   SETBIT          ;
00A6 020E                M                movf    Preg_a,W        ;
00A7 0E6D                M                andlw   OR_a3           ;
00A8 0743                M                btfss   status,bit2     ;
00A9 0AB1                M                goto    SETBIT          ;
                         M
00AA 020F                M                movf    Preg_b,W        ;
00AB 0E79                M                andlw   OR_b3           ;
00AC 0743                M                btfss   status,bit2     ;
00AD 0AB1                M                goto    SETBIT          ;
                         M
00AE 0210                M                movf    Preg_c,W        ;
```


© 1997 Microchip Technology Inc.

```
00AF 0E00                M                andlw   OR_c3           ;
00B0 0743                M                btfss   status,bit2     ;
00B1 056D                M SETBIT         bsf     Y_reg,bit3      ;
                       00208              EVAL_Y  OR_a4,OR_b4,OR_c4,bit4
  0000                   M                LOCAL   SETBIT          ;
00B2 020E                M                movf    Preg_a,W        ;
00B3 0E45                M                andlw   OR_a4           ;
00B4 0743                M                btfss   status,bit2     ;
00B5 0ABD                M                goto    SETBIT          ;
                         M
00B6 020F                M                movf    Preg_b,W        ;
00B7 0EFD                M                andlw   OR_b4           ;
00B8 0743                M                btfss   status,bit2     ;
00B9 0ABD                M                goto    SETBIT          ;
                         M
00BA 0210                M                movf    Preg_c,W        ;
00BB 0E00                M                andlw   OR_c4           ;
00BC 0743                M                btfss   status,bit2     ;
00BD 058D                M SETBIT         bsf     Y_reg,bit4      ;
                       00209              EVAL_Y  OR_a5,OR_b5,OR_c5,bit5
  0000                   M                LOCAL   SETBIT          ;
00BE 020E                M                movf    Preg_a,W        ;
00BF 0E71                M                andlw   OR_a5           ;
00C0 0743                M                btfss   status,bit2     ;
00C1 0AC9                M                goto    SETBIT          ;
                         M
Preg_b,W
00C3 0EDF                M                andlw   OR_b5           ;
00C4 0743                M                btfss   status,bit2     ;
00C5 0AC9                M                goto    SETBIT          ;
                         M
00C6 0210                M                movf    Preg_c,W        ;
00C7 0E00                M                andlw   OR_c5           ;
00C8 0743                M                btfss   status,bit2     ;
00C9 05AD                M SETBIT         bsf     Y_reg,bit5      ;
                       00210              EVAL_Y  OR_a6,OR_b6,OR_c6,bit6
  0000                   M                LOCAL   SETBIT          ;
00CA 020E                M                movf    Preg_a,W        ;
00CB 0E7C                M                andlw   OR_a6           ;
00CC 0743                M                btfss   status,bit2     ;
00CD 0AD5                M                goto    SETBIT          ;
                         M
00CE 020F                M                movf    Preg_b,W        ;
00CF 0EEF                M                andlw   OR_b6           ;
00D0 0743                M                btfss   status,bit2     ;
00D1 0AD5                M                goto    SETBIT          ;
                         M
00D2 0210                M                movf    Preg_c,W        ;
00D3 0E00                M                andlw   OR_c6           ;
00D4 0743                M                btfss   status,bit2     ;
00D5 05CD                M SETBIT  bsf                Y_reg,bit6  ;
                       00211              EVAL_Y  OR_a7,OR_b7,OR_c7,bit7
  0000                   M                LOCAL   SETBIT          ;
00D6 020E                M                movf    Preg_a,W        ;
00D7 0E00                M                andlw   OR_a7           ;
00D8 0743                M                btfss   status,bit2     ;
00D9 0AE1                M                goto    SETBIT          ;
                         M
00DA 020F                M                movf    Preg_b,W        ;
00DB 0E00                M                andlw   OR_b7           ;
00DC 0743                M                btfss   status,bit2     ;
00DD 0AE1                M                goto    SETBIT          ;
                         M
00DE 0210                M                movf    Preg_c,W        ;
00DF 0E00                M                andlw   OR_c7           ;
00E0 0743                M                btfss   status,bit2     ;
```


© 1997 Microchip Technology Inc.

```
00E1 05ED                M SETBIT  bsf             Y_reg,bit7      ;
                    00212
                    00213 ; Y_reg now contains 8 output values:
00E2 0040           00214 wr_out  clrw                            ;
00E3 0007           00215                 tris    7               ; port_c = output
00E4 020D           00216                 movf    Y_reg,W         ;
00E5 0027           00217                 movwf   port_c          ; Y_reg -> port_c
00E6 0800           00218                 retlw   0               ;
                    00219
00E7 0000           00220 ZZZ             nop
                    00221
                    00222         END


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXX-------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   233
Program Memory Words Free:   279


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: PLA IMPLEMENTATION: CODE EFFICIENT APPROACH

```
MPASM 01.40 Released            PLA1B.ASM   1-16-1997  12:29:06         PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ; pla1b.asm :
                     00005 ; This procedure implements a simple AND-OR PLA with:
                     00006 ;
                     00007 ;       8  inputs         := A7 A6 A5 A4 A3 A2 A1 A0
                     00008 ;       24 product terms  := P23 P22 ..... P0
                     00009 ;       8  outputs        := Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
                     00010 ;
                     00011 ; The eight inputs are assumed to be connected to PORT RB such that
                     00012 ; RB0 = A0, RB1 = A1, ... , RB7 = A7.
                     00013 ; The outputs are programmed to appear on port RC such that
                     00014 ; RC0 = Y0, RC1 = Y1, ... , RC7 = Y7.
                     00015 ;
                     00016 ; This implementation optimizes program memory usage over
                     00017 ; speed
                     00018 ;
                     00019 ;       Program:          PLA1B.ASM
                     00020 ;       Revision Date:
                     00021 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00022 ;
                     00023 ;*****************************************************************
                     00024 ;
                     00025 ; define RAM locations used:
                     00026 ;
0000000C             00027 input   equ     d'12'   ; RAM location 12 holds input
0000000D             00028 Y_reg   equ     d'13'   ; holds output result
                     00029
0000000E             00030 Preg_a  equ     d'14'   ; Product terms P0 to P7. Preg_a<0> = P0
0000000F             00031 Preg_b  equ     d'15'   ; Product terms P8 to P15. Preg_b<0> = P8
00000010             00032 Preg_c  equ     d'16'   ; Product terms P16 to P23. Preg_c<0> = P16
                     00033
00000012             00034 Pn_x    equ     d'18'   ;
00000013             00035 Pn_a    equ     d'19'   ;
00000014             00036 OR_a    equ     d'20'   ;
00000015             00037 OR_b    equ     d'21'   ;
00000016             00038 OR_c    equ     d'22'   ;
                     00039
                     00040 ; define some constants and file addresses:
                     00041 ;
00000000             00042 bit0    equ     0       ;
00000001             00043 bit1    equ     1       ;
00000002             00044 bit2    equ     2       ;
00000003             00045 bit3    equ     3       ;
00000004             00046 bit4    equ     4       ;
00000005             00047 bit5    equ     5       ;
00000006             00048 bit6    equ     6       ;
00000007             00049 bit7    equ     7       ;
                     00050 ;
00000003             00051 status  equ           ;
00000006             00052 port_b  equ     6       ;
00000007             00053 port_c  equ     7       ;
                     00054 ;
                     00055 ; define the AND plane programming variables:
```

```
                    00056 ;
00000000            00057 P0_x    equ     b'00000000'     ;
0000000F            00058 P0_a    equ     b'00001111'     ;
00000001            00059 P1_x    equ     b'00000001'     ;
0000000F            00060 P1_a    equ     b'00001111'     ;
00000002            00061 P2_x    equ     b'00000010'     ;
0000000F            00062 P2_a    equ     b'00001111'     ;
00000003            00063 P3_x    equ     b'00000011'     ;
0000000F            00064 P3_a    equ     b'00001111'     ;
00000004            00065 P4_x    equ     b'00000100'     ;
0000000F            00066 P4_a    equ     b'00001111'     ;
00000005            00067 P5_x    equ     b'00000101'     ;
0000000F            00068 P5_a    equ     b'00001111'     ;
00000006            00069 P6_x    equ     b'00000110'     ;
0000000F            00070 P6_a    equ     b'00001111'     ;
00000007            00071 P7_x    equ     b'00000111'     ;
0000000F            00072 P7_a    equ     b'00001111'     ;
00000008            00073 P8_x    equ     b'00001000'     ;
0000000F            00074 P8_a    equ     b'00001111'     ;
00000009            00075 P9_x    equ     b'00001001'     ;
0000000F            00076 P9_a    equ     b'00001111'     ;
0000000A            00077 P10_x   equ     b'00001010'     ;
0000000F            00078 P10_a   equ     b'00001111'     ;
0000000B            00079 P11_x   equ     b'00001011'     ;
0000000F            00080 P11_a   equ     b'00001111'     ;
0000000C            00081 P12_x   equ     b'00001100'     ;
0000000F            00082 P12_a   equ     b'00001111'     ;
0000000D            00083 P13_x   equ     b'00001101'     ;
0000000F            00084 P13_a   equ     b'00001111'     ;
0000000E            00085 P14_x   equ     b'00001110'     ;
0000000F            00086 P14_a   equ     b'00001111'     ;
0000000F            00087 P15_x   equ     b'00001111'     ;
0000000F            00088 P15_a   equ     b'00001111'     ;
00000000            00089 P16_x   equ     b'00000000'     ;
00000000            00090 P16_a   equ     b'00000000'     ;
00000000            00091 P17_x   equ     b'00000000'     ;
00000000            00092 P17_a   equ     b'00000000'     ;
00000000            00093 P18_x   equ     b'00000000'     ;
00000000            00094 P18_a   equ     b'00000000'     ;
00000000            00095 P19_x   equ     b'00000000'     ;
00000000            00096 P19_a   equ     b'00000000'     ;
00000000            00097 P20_x   equ     b'00000000'     ;
00000000            00098 P20_a   equ     b'00000000'     ;
00000000            00099 P21_x   equ     b'00000000'     ;
00000000            00100 P21_a   equ     b'00000000'     ;
00000000            00101 P22_x   equ     b'00000000'     ;
00000000            00102 P22_a   equ     b'00000000'     ;
00000000            00103 P23_x   equ     b'00000000'     ;
00000000            00104 P23_a   equ     b'00000000'     ;
                    00105
                    00106 ; define OR plane programming variables:
                    00107
000000ED            00108 OR_a0   equ     b'11101101'     ; for output Y0
000000D7            00109 OR_b0   equ     b'11010111'     ;
00000000            00110 OR_c0   equ     b'00000000'     ;
0000009F            00111 OR_a1   equ     b'10011111'     ; for output Y1
00000027            00112 OR_b1   equ     b'00100111'     ;
00000000            00113 OR_c1   equ     b'00000000'     ;
000000FB            00114 OR_a2   equ     b'11111011'     ; for output Y2
0000002F            00115 OR_b2   equ     b'00101111'     ;
00000000            00116 OR_c2   equ     b'00000000'     ;
0000006D            00117 OR_a3   equ     b'01101101'     ; for output Y3
00000079            00118 OR_b3   equ     b'01111001'     ;
00000000            00119 OR_c3   equ     b'00000000'     ;
00000045            00120 OR_a4   equ     b'01000101'     ; for output Y4
000000FD            00121 OR_b4   equ     b'11111101'     ;
```


© 1997 Microchip Technology Inc.

```
  00000000          00122 OR_c4   equ     b'00000000'             ;
  00000071          00123 OR_a5   equ     b'01110001'             ; for output Y5
  000000DF          00124 OR_b5   equ     b'11011111'             ;
  00000000          00125 OR_c5   equ     b'00000000'             ;
  0000007C          00126 OR_a6   equ     b'01111100'             ; for output Y6
  000000EF          00127 OR_b6   equ     b'11101111'             ;
  00000000          00128 OR_c6   equ     b'00000000'             ;
  00000000          00129 OR_a7   equ     b'00000000'             ; for output Y7
  00000000          00130 OR_b7   equ     b'00000000'             ;
  00000000          00131 OR_c7   equ     b'00000000'             ;
                    00132
                    00133
01FF                00134                 org     01ffh           ;
01FF 0A00           00135 begin   goto    main                    ;
                    00136
0000                00137                 org     000h            ;
                    00138 ; define macro to evaluate 1 product (AND) term:
                    00139 ;
0000 090B           00140 main            call    pla88           ;
0001 0A00           00141                 goto    main            ;
                    00142 ;
                    00143
                    00144 EVAL_P  MACRO   Preg_x,bit_n,Pn_x,Pn_a
                    00145                 movf    input,W         ;
                    00146                 xorlw   Pn_x            ;
                    00147                 andlw   Pn_a            ;
                    00148                 btfsc   status,bit2     ; skip if zero bit not set
                    00149                 bsf     Preg_x,bit_n    ; product term = 1
                    00150                 ENDM
                    00151
                    00152
                    00153 ; define macro to load OR term constants:
                    00154 ;
                    00155 EVAL_Y  MACRO   OR_an,OR_bn,OR_cn,bit_n
                    00156                 movlw   OR_an   ; load constants
                    00157                 movwf   OR_a            ;
                    00158                 movlw   OR_bn           ;
                    00159                 movwf   OR_b            ;
                    00160                 movlw   OR_cn           ;
                    00161                 movwf   OR_c            ;
                    00162                 call    EVAL1           ;
                    00163                 btfss   status,bit2     ;
                    00164                 bsf     Y_reg,bit_n     ;
                    00165                 ENDM
                    00166
                    00167 ; define procedure to evaluate 1 output (OR) term:
                    00168 ;
0002 020E           00169 EVAL1   movf            Preg_a,W        ;
0003 0174           00170                 andwf   OR_a,1          ;
                    00171
0004 020F           00172                 movf    Preg_b,W        ;
0005 0175           00173                 andwf   OR_b,1          ;
                    00174
0006 0210           00175                 movf    Preg_c,W        ;
0007 0156           00176                 andwf   OR_c,W          ;
                    00177
0008 0114           00178                 iorwf   OR_a,W          ;
0009 0115           00179                 iorwf   OR_b,W          ;
000A 0800           00180                 retlw   0               ;  W = 1 implies Yn = 1
                    00181
                    00182 ; now the  PLA evaluation procedure:
                    00183 ;
000B 0CFF           00184 pla88   movlw   0ffh                    ;
000C 0006           00185                 tris    6               ; port_b = input
000D 0206           00186                 movf    port_b,W        ; read input
000E 002C           00187                 movwf   input           ; store input in a register
```

```
000F 006E          00188                clrf    Preg_a          ; clear Product register a
0010 006F          00189                clrf    Preg_b          ; clear Product register b
0011 0070          00190                clrf    Preg_c          ; clear Product register c
0012 006D          00191                clrf    Y_reg           ; clear output register
                   00192
                   00193 and_p1  EVAL_P  Preg_a,bit0,P0_x,P0_a
0013 020C             M                 movf    input,W         ;
0014 0F00             M                 xorlw   P0_x            ;
0015 0E0F             M                 andlw   P0_a            ;
0016 0643             M                 btfsc   status,bit2     ; skip if zero bit not set
0017 050E             M                 bsf     Preg_a,bit0     ; product term = 1
                   00194                EVAL_P  Preg_a,bit1,P1_x,P1_a
0018 020C             M                 movf    input,W         ;
0019 0F01             M                 xorlw   P1_x            ;
001A 0E0F             M                 andlw   P1_a            ;
001B 0643             M                 btfsc   status,bit2     ; skip if zero bit not set
001C 052E             M                 bsf     Preg_a,bit1     ; product term = 1
                   00195                EVAL_P  Preg_a,bit2,P2_x,P2_a
001D 020C             M                 movf    input,W         ;
001E 0F02             M                 xorlw   P2_x            ;
001F 0E0F             M                 andlw   P2_a            ;
0020 0643             M                 btfsc   status,bit2     ; skip if zero bit not set
0021 054E             M                 bsf     Preg_a,bit2     ; product term = 1
                   00196                EVAL_P  Preg_a,bit3,P3_x,P3_a
0022 020C             M                 movf    input,W         ;
0023 0F03             M                 xorlw   P3_x            ;
0024 0E0F             M                 andlw   P3_a            ;
0025 0643             M                 btfsc   status,bit2     ; skip if zero bit not set
0026 056E             M                 bsf     Preg_a,bit3     ; product term = 1
                   00197                EVAL_P  Preg_a,bit4,P4_x,P4_a
0027 020C             M                 movf    input,W         ;
0028 0F04             M                 xorlw   P4_x            ;
0029 0E0F             M                 andlw   P4_a            ;
002A 0643             M                 btfsc   status,bit2     ; skip if zero bit not set
002B 058E             M                 bsf     Preg_a,bit4     ; product term = 1
                   00198                EVAL_P  Preg_a,bit5,P5_x,P5_a
002C 020C             M                 movf    input,W         ;
002D 0F05             M                 xorlw   P5_x            ;
002E 0E0F             M                 andlw   P5_a            ;
002F 0643             M                 btfsc   status,bit2     ; skip if zero bit not set
0030 05AE             M                 bsf     Preg_a,bit5     ; product term = 1
                   00199                EVAL_P  Preg_a,bit6,P6_x,P6_a
0031 020C             M                 movf    input,W         ;
0032 0F06             M                 xorlw   P6_x            ;
0033 0E0F             M                 andlw   P6_a            ;
0034 0643             M                 btfsc   status,bit2     ; skip if zero bit not set
0035 05CE             M                 bsf     Preg_a,bit6     ; product term = 1
                   00200                EVAL_P  Preg_a,bit7,P7_x,P7_a
0036 020C             M                 movf    input,W         ;
0037 0F07             M                 xorlw   P7_x            ;
0038 0E0F             M                 andlw   P7_a            ;
0039 0643             M                 btfsc   status,bit2     ; skip if zero bit not set
003A 05EE             M                 bsf     Preg_a,bit7     ; product term = 1
                   00201
                   00202                EVAL_P  Preg_b,bit0,P8_x,P8_a
003B 020C             M                 movf    input,W         ;
003C 0F08             M                 xorlw   P8_x            ;
003D 0E0F             M                 andlw   P8_a            ;
003E 0643             M                 btfsc   status,bit2     ; skip if zero bit not set
003F 050F             M                 bsf     Preg_b,bit0     ; product term = 1
                   00203                EVAL_P  Preg_b,bit1,P9_x,P9_a
0040 020C             M                 movf    input,W         ;
0041 0F09             M                 xorlw   P9_x            ;
0042 0E0F             M                 andlw   P9_a            ;
0043 0643             M                 btfsc   status,bit2     ; skip if zero bit not set
0044 052F             M                 bsf     Preg_b,bit1     ; product term = 1
```


© 1997 Microchip Technology Inc.

```
                    00204                EVAL_P  Preg_b,bit2,P10_x,P10_a
0045 020C               M                movf    input,W          ;
0046 0F0A               M                xorlw   P10_x            ;
0047 0E0F               M                andlw   P10_a            ;
0048 0643               M                btfsc   status,bit2      ; skip if zero bit not set
0049 054F               M                bsf     Preg_b,bit2      ; product term = 1
                    00205                EVAL_P  Preg_b,bit3,P11_x,P11_a
004A 020C               M                movf    input,W          ;
004B 0F0B               M                xorlw   P11_x            ;
004C 0E0F               M                andlw   P11_a            ;
004D 0643               M                btfsc   status,bit2      ; skip if zero bit not set
004E 056F               M                bsf     Preg_b,bit3      ; product term = 1
                    00206                EVAL_P  Preg_b,bit4,P12_x,P12_a
004F 020C               M                movf    input,W          ;
0050 0F0C               M                xorlw   P12_x            ;
0051 0E0F               M                andlw   P12_a            ;
0052 0643               M                btfsc   status,bit2      ; skip if zero bit not set
0053 058F               M                bsf     Preg_b,bit4      ; product term = 1
                    00207                EVAL_P  Preg_b,bit5,P13_x,P13_a
0054 020C               M                movf    input,W          ;
0055 0F0D               M                xorlw   P13_x            ;
0056 0E0F               M                andlw   P13_a            ;
0057 0643               M                btfsc   status,bit2      ; skip if zero bit not set
0058 05AF               M                bsf     Preg_b,bit5      ; product term = 1
                    00208                EVAL_P  Preg_b,bit6,P14_x,P14_a
0059 020C               M                movf    input,W          ;
005A 0F0E               M                xorlw   P14_x            ;
005B 0E0F               M                andlw   P14_a            ;
005C 0643               M                btfsc   status,bit2      ; skip if zero bit not set
005D 05CF               M                bsf     Preg_b,bit6      ; product term = 1
                    00209                EVAL_P  Preg_b,bit7,P15_x,P15_a
005E 020C               M                movf    input,W          ;
005F 0F0F               M                xorlw   P15_x            ;
0060 0E0F               M                andlw   P15_a            ;
0061 0643               M                btfsc   status,bit2      ; skip if zero bit not set
0062 05EF               M                bsf     Preg_b,bit7      ; product term = 1
                    00210
                    00211                EVAL_P  Preg_c,bit0,P16_x,P16_a
0063 020C               M                movf    input,W          ;
0064 0F00               M                xorlw   P16_x            ;
0065 0E00               M                andlw   P16_a            ;
0066 0643               M                btfsc   status,bit2      ; skip if zero bit not set
0067 0510               M                bsf     Preg_c,bit0      ; product term = 1
                    00212                EVAL_P  Preg_c,bit1,P17_x,P17_a
0068 020C               M                movf    input,W          ;
0069 0F00               M                xorlw   P17_x            ;
006A 0E00               M                andlw   P17_a            ;
006B 0643               M                btfsc   status,bit2      ; skip if zero bit not set
006C 0530               M                bsf     Preg_c,bit1      ; product term = 1
                    00213                EVAL_P  Preg_c,bit2,P18_x,P18_a
006D 020C               M                movf    input,W          ;
006E 0F00               M                xorlw   P18_x            ;
006F 0E00               M                andlw   P18_a            ;
0070 0643               M                btfsc   status,bit2      ; skip if zero bit not set
0071 0550               M                bsf     Preg_c,bit2      ; product term = 1
                    00214                EVAL_P  Preg_c,bit3,P19_x,P19_a
0072 020C               M                movf    input,W          ;
0073 0F00               M                xorlw   P19_x            ;
0074 0E00               M                andlw   P19_a            ;
0075 0643               M                btfsc   status,bit2      ; skip if zero bit not set
0076 0570               M                bsf     Preg_c,bit3      ; product term = 1
                    00215                EVAL_P  Preg_c,bit4,P20_x,P20_a
0077 020C               M                movf    input,W          ;
0078 0F00               M                xorlw   P20_x            ;
0079 0E00               M                andlw   P20_a            ;
007A 0643               M                btfsc   status,bit2      ; skip if zero bit not set
```


© 1997 Microchip Technology Inc.

```
007B 0590          M            bsf     Preg_c,bit4     ; product term = 1
                 00216          EVAL_P  Preg_c,bit5,P21_x,P21_a
007C 020C          M            movf    input,W         ;
007D 0F00          M            xorlw   P21_x           ;
007E 0E00          M            andlw   P21_a           ;
007F 0643          M            btfsc   status,bit2     ; skip if zero bit not set
0080 05B0          M            bsf     Preg_c,bit5     ; product term = 1
                 00217          EVAL_P  Preg_c,bit6,P22_x,P22_a
0081 020C          M            movf    input,W         ;
0082 0F00          M            xorlw   P22_x           ;
0083 0E00          M            andlw   P22_a           ;
0084 0643          M            btfsc   status,bit2     ; skip if zero bit not set
0085 05D0          M            bsf     Preg_c,bit6     ; product term = 1
                 00218          EVAL_P  Preg_c,bit7,P23_x,P23_a
0086 020C          M            movf    input,W         ;
0087 0F00          M            xorlw   P23_x           ;
0088 0E00          M            andlw   P23_a           ;
0089 0643          M            btfsc   status,bit2     ; skip if zero bit not set
008A 05F0          M            bsf     Preg_c,bit7     ; product term = 1
                 00219
                 00220 or_p1    EVAL_Y   OR_a0,OR_b0,OR_c0,bit0
008B 0CED          M            movlw   OR_a0           ; load constants
008C 0034          M            movwf   OR_a            ;
008D 0CD7          M            movlw   OR_b0           ;
008E 0035          M            movwf   OR_b            ;
008F 0C00          M            movlw   OR_c0           ;
0090 0036          M            movwf   OR_c            ;
0091 0902          M            call    EVAL1           ;
0092 0743          M            btfss   status,bit2     ;
0093 050D          M            bsf     Y_reg,bit0      ;
                 00221          EVAL_Y  OR_a1,OR_b1,OR_c1,bit1
0094 0C9F          M            movlw   OR_a1           ; load constants
0095 0034          M            movwf   OR_a            ;
0096 0C27          M            movlw   OR_b1           ;
0097 0035          M            movwf   OR_b            ;
0098 0C00          M            movlw   OR_c1           ;
0099 0036          M            movwf   OR_c            ;
009A 0902          M            call    EVAL1           ;
009B 0743          M            btfss   status,bit2     ;
009C 052D          M            bsf     Y_reg,bit1      ;
                 00222          EVAL_Y  OR_a2,OR_b2,OR_c2,bit2
009D 0CFB          M            movlw   OR_a2           ; load constants
009E 0034          M            movwf   OR_a            ;
009F 0C2F          M            movlw   OR_b2           ;
00A0 0035          M            movwf   OR_b            ;
00A1 0C00          M            movlw   OR_c2           ;
00A2 0036          M            movwf   OR_c            ;
00A3 0902          M            call    EVAL1           ;
00A4 0743          M            btfss   status,bit2     ;
00A5 054D          M            bsf     Y_reg,bit2      ;
                 00223          EVAL_Y  OR_a3,OR_b3,OR_c3,bit3
00A6 0C6D          M            movlw   OR_a3           ; load constants
00A7 0034          M            movwf   OR_a            ;
00A8 0C79          M            movlw   OR_b3           ;
00A9 0035          M            movwf   OR_b            ;
00AA 0C00          M            movlw   OR_c3           ;
00AB 0036          M            movwf   OR_c            ;
00AC 0902          M            call    EVAL1           ;
00AD 0743          M            btfss   status,bit2     ;
00AE 056D          M            bsf     Y_reg,bit3      ;
                 00224          EVAL_Y  OR_a4,OR_b4,OR_c4,bit4
00AF 0C45          M            movlw   OR_a4           ; load constants
00B0 0034          M            movwf   OR_a            ;
00B1 0CFD          M            movlw   OR_b4           ;
00B2 0035          M            movwf   OR_b            ;
00B3 0C00          M            movlw   OR_c4           ;
```


© 1997 Microchip Technology Inc.

```
00B4 0036               M                movwf   OR_c            ;
00B5 0902               M                call    EVAL1           ;
00B6 0743               M                btfss   status,bit2     ;
00B7 058D               M                bsf     Y_reg,bit4      ;
                      00225              EVAL_Y  OR_a5,OR_b5,OR_c5,bit5
00B8 0C71               M                movlw   OR_a5           ; load constants
00B9 0034               M                movwf   OR_a            ;
00BA 0CDF               M                movlw   OR_b5           ;
00BB 0035               M                movwf   OR_b            ;
00BC 0C00               M                movlw   OR_c5           ;
00BD 0036               M                movwf   OR_c            ;
00BE 0902               M                call    EVAL1           ;
00BF 0743               M                btfss   status,bit2     ;
00C0 05AD               M                bsf     Y_reg,bit5      ;
                      00226              EVAL_Y  OR_a6,OR_b6,OR_c6,bit6
00C1 0C7C               M                movlw   OR_a6           ; load constants
00C2 0034               M                movwf   OR_a            ;
00C3 0CEF               M                movlw   OR_b6           ;
00C4 0035               M                movwf   OR_b            ;
00C5 0C00               M                movlw   OR_c6           ;
00C6 0036               M                movwf   OR_c            ;
00C7 0902               M                call    EVAL1           ;
00C8 0743               M                btfss   status,bit2     ;
00C9 05CD               M                bsf     Y_reg,bit6      ;
                      00227              EVAL_Y  OR_a7,OR_b7,OR_c7,bit7
00CA 0C00               M                movlw   OR_a7           ; load constants
00CB 0034               M                movwf   OR_a            ;
00CC 0C00               M                movlw   OR_b7           ;
00CD 0035               M                movwf   OR_b            ;
00CE 0C00               M                movlw   OR_c7           ;
00CF 0036               M                movwf   OR_c            ;
00D0 0902               M                call    EVAL1           ;
00D1 0743               M                btfss   status,bit2     ;
00D2 05ED               M                bsf     Y_reg,bit7      ;
                      00228
                      00229 ; Y_reg now contains 8 output values:
00D3 0040             00230 wr_out  clrw                         ;
00D4 0007             00231                 tris    7            ; port_c = output
00D5 020D             00232                 movf    Y_reg,W      ;
00D6 0027             00233                 movwf   port_c       ; Y_reg -> port_c
00D7 0800             00234                 retlw   0            ;
                      00235
00D8 0000             00236 ZZZ             nop
                      00237
                      00238         END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXX------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   218
Program Memory Words Free:   294


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Implementing Ohmmeter/Temperature Sensor

Author: *Doug Cox*
Microchip Technology, Inc.

INTRODUCTION

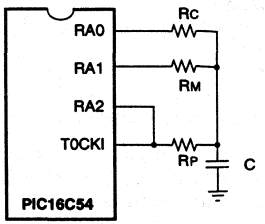
This application note describes a method for implementing an ohmmeter or resistance type temperature sensor using the PIC16C5X series of microcontrollers. The ohmmeter requires only two external components and is software and hardware configurable for resistance measurement with resolutions from 6-bits to 10-bits with measurement times of 250 µs (6-bits at 8 MHz) or longer. This method uses a software calibration technique that compensates for voltage, time, and temperature drift as well as component errors. PIC16C5X microcontrollers are ideal for simple analog applications because:

- Very low cost
- Few external components required
- Fully programmable. PIC16C5X Microcontrollers are offered as One Time Programmable (OTP) EPROM devices.
- Available off the shelf from distributors
- Calibration in software for improved measurement accuracy
- Power savings using PIC16C5X's Sleep mode.
- PIC16C5X's output pins have large, current source/sink capability to drive LED's directly.

THEORY OF OPERATION

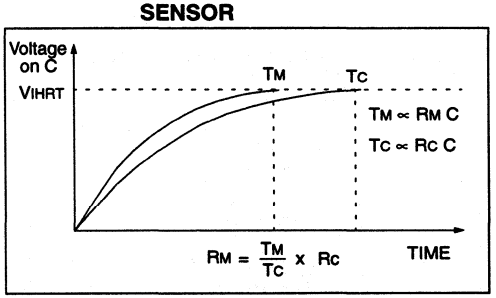
This application uses a capacitive charging circuit (Figure 1) to convert resistance to time, which can be easily measured using a microcontroller. First, a reference voltage (usually VDD) is applied to a calibration resistor, Rc. The capacitor C is charged up until the threshold on the chip input trips. This generates a software calibration value that is used to calibrate out most circuit errors, including inaccuracies in the capacitor, changes in the input threshold voltage and temperature variations. After C is discharged, the reference voltage is applied to the resistance to be measured (or thermistor). The time to trip the threshold is then measured and compared to the calibration value to determine the actual resistance (Figure 2). In the temperature sensing mode, the temperature is calculated using a lookup table.

FIGURE 1: OHMMETER/TEMPERATURE SENSOR



RP is a small resistor (100-200Ω) which limits peak current through RA2 while discharging, or through RA2 or T0CKI in the event of an Electro-Static Discharge (ESD) or Electrical Over Stress (EOS) related breakdown.

FIGURE 2: OHMMETER/TEMPERATURE SENSOR



© 1997 Microchip Technology Inc.

CIRCUIT CONFIGURATION

The values of Rc and C are selected based upon the number of bits of resolution required. Rc should be approximately one half the largest value resistance to be measured and:

$$C = \frac{-T}{R_M \bullet IN\left(1 - \frac{V_T}{V_R}\right)}$$

Where:

V_R = Reference voltage
T = Time to do the number of bits of resolution desired
V_T = Threshold voltage of the PIC16C5X input being used
R_M = Maximum resistance value to be measured

Actual value for C should be slightly smaller than calculated to ensure that the PIC16C5X does not overcount during the measurement.

For example use R_M = 200k for 8-bit resolution with an 8 MHz clock, V_R = 5V, V_T = 3V, Rc = 100k and 6 instruction cycles per count:

T = 256 counts • 1/8 MHz • 4 clocks/instruction • 6 instructions/count = 768 µs
C = 4200 pF [Use 3900 pF]

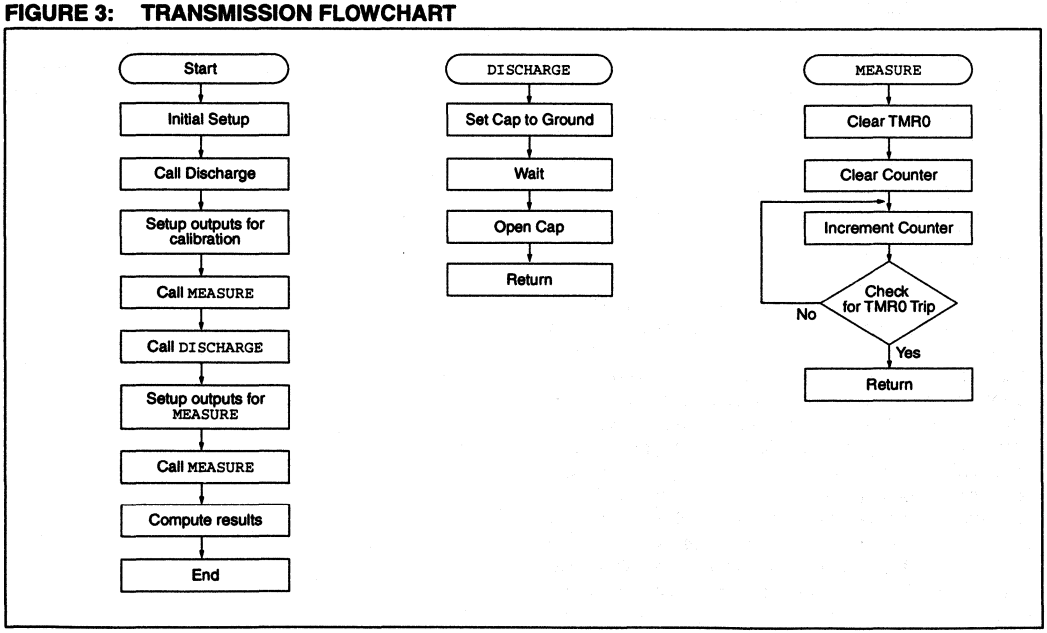
CIRCUIT PERFORMANCE

The calibration cycle removes all first order errors (offset, gain, C inaccuracy, power supply voltage and temperature) except R absolute accuracy. A low drift resistor should be selected for R and its value stored in software to reduce measurement errors. Other error sources are I/O pin leakage, resistor and capacitor non-linearities, input threshold uncertainty, and time measurement uncertainty (± one instruction cycle time). Measured performance shows the ohmmeter to be accurate within ±1% over one decade.

Example

The assembly code implementing the circuit of Figure 1 is listed in Appendix A. This code measures time up to 16-bits (65535 measure cycles) and calculates the results using 16-bit multiply and divide subroutines. In actual applications, it is more efficient to use 8-bit measurements if application accuracies permit. The math code will be substantially reduced and measurement time is reduced by the simpler code and shorter count.

FIGURE 3: TRANSMISSION FLOWCHART



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: OHMETER SOURCE CODE

```
MPASM 01.40 Released          OHMETER.ASM   1-16-1997  12:31:45         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         TITLE  'OHMETER/TEMPERATURE SENSING PROGRAM REV 3-29-90'
                     00002         LIST   P=16C54
                     00003 ;
                     00004 ;*****************************************************************
                     00005 ;
                     00006 ;       Program:          OHMETER.ASM
                     00007 ;       Revision Date:
                     00008 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00009 ;
                     00010 ;*****************************************************************
                     00011 ;
  00000008           00012 ACCA    EQU     8
  0000000A           00013 ACCB    EQU     0A
  0000000C           00014 ACCC    EQU     0C
  0000000E           00015 ACCD    EQU     0E
  00000010           00016 ACCE    EQU     10
  00000012           00017 TCAL    EQU     12
  00000014           00018 TEMP    EQU     14
                     00019 ;
  00000001           00020 F       EQU     1
                     00021 ;
  0000002F           00022 RCALMS  EQU     2F            ;RCAL MSB VALUE IN HEX
  0000003C           00023 RCALLS  EQU     3C            ;RCAL LSB VALUE IN HEX
                     00024
01FF                 00025         ORG     1FF
01FF 0A58            00026         GOTO    OHMS
0000                 00027         ORG     0
                     00028
0000 0209            00029 MADD    MOVF    ACCA+1,W
0001 01EB            00030         ADDWF   ACCB+1, F     ;ADD LSB
0002 0603            00031         BTFSC   3,0           ;ADD IN CARRY
0003 02AA            00032         INCF    ACCB, F
0004 0208            00033         MOVF    ACCA,W
0005 01EA            00034         ADDWF   ACCB, F       ;ADD MSB
0006 0800            00035         RETLW   0
0007 0000            00036         NOP
                     00037
0008 0915            00038 MPY     CALL    SETUP         ;RESULTS IN B(16 MSB'S) AND C(16 LSB'S)
0009 032E            00039 MLOOP   RRF     ACCD, F       ;ROTATE D RIGHT
000A 032F            00040         RRF     ACCD+1, F
000B 0603            00041         SKPNC                 ;NEED TO ADD?
000C 0900            00042         CALL    MADD
000D 032A            00043         RRF     ACCB, F
000E 032B            00044         RRF     ACCB+1, F
000F 032C            00045         RRF     ACCC, F
0010 032D            00046         RRF     ACCC+1, F
0011 02F4            00047         DECFSZ  TEMP, F       ;LOOP UNTIL ALL BITS CHECKED
0012 0A09            00048         GOTO    MLOOP
0013 0800            00049         RETLW   0
                     00050
0014 0000            00051         NOP
0015 0C10            00052 SETUP   MOVLW   10
0016 0034            00053         MOVWF   TEMP
0017 020A            00054         MOVF    ACCB,W        ;MOVE B TO D
0018 002E            00055         MOVWF   ACCD
```

```
0019 020B           00056          MOVF    ACCB+1,W
001A 002F           00057          MOVWF   ACCD+1
001B 020C           00058          MOVF    ACCC,W
001C 0030           00059          MOVWF   ACCE
001D 020D           00060          MOVF    ACCC+1,W
001E 0031           00061          MOVWF   ACCE+1
001F 006A           00062          CLRF    ACCB
0020 006B           00063          CLRF    ACCB+1
0021 0800           00064          RETLW   0
                    00065
0022 0000           00066          NOP
0023 0915           00067 DIV      CALL    SETUP
0024 0C20           00068          MOVLW   20
0025 0034           00069          MOVWF   TEMP
0026 006C           00070          CLRF    ACCC
0027 006D           00071          CLRF    ACCC+1
0028 0403           00072 DLOOP    CLRC
0029 0371           00073          RLF     ACCE+1, F
002A 0370           00074          RLF     ACCE, f
002B 036F           00075          RLF     ACCD+1, F
002C 036E           00076          RLF     ACCD, F
002D 036D           00077          RLF     ACCC+1, F
002E 036C           00078          RLF     ACCC, F
002F 0208           00079          MOVF    ACCA,W
0030 008C           00080          SUBWF   ACCC,W         ;CHECK IF A>C
0031 0743           00081          SKPZ
0032 0A35           00082          GOTO    NOCHK
0033 0209           00083          MOVF    ACCA+1,W
0034 008D           00084          SUBWF   ACCC+1,W       ;IF MSB EQUAL THEN CHECK LSB
0035 0703           00085 NOCHK    SKPC                   ;CARRY SET IF C>A
0036 0A3E           00086          GOTO    NOGO
0037 0209           00087          MOVF    ACCA+1,W       ;C-A INTO C
0038 00AD           00088          SUBWF   ACCC+1, F
0039 0703           00089          BTFSS   3,0
003A 00EC           00090          DECF    ACCC, F
003B 0208           00091          MOVF    ACCA,W
003C 00AC           00092          SUBWF   ACCC, F
003D 0503           00093          SETC                   ;SHIFT A 1 INTO B (RESULT)
003E 036B           00094 NOGO     RLF     ACCB+1, F
003F 036A           00095          RLF     ACCB, F
0040 02F4           00096          DECFSZ  TEMP, F        ;LOOP UNTILL ALL BITS CHECKED
0041 0A28           00097          GOTO    DLOOP
0042 0800           00098          RETLW   0
                    00099
0043 0C0B           00100 DSCHRG   MOVLW   B'00001011'    ;ACTIVATE RA2
0044 0005           00101          TRIS    5
0045 0CFF           00102          MOVLW   0FF
0046 0034           00103          MOVWF   TEMP
0047 02F4           00104 LOOP     DECFSZ  TEMP, F        ;WAIT
0048 0A47           00105          GOTO    LOOP
0049 0C0F           00106          MOVLW   B'00001111'    ;ALL OUTPUTS OFF
004A 0005           00107          TRIS    5
004B 0800           00108          RETLW   0
                    00109
004C 0061           00110 M_TIME   CLRF    1              ;CLEAR TMR0
004D 0069           00111          CLRF    ACCA+1
004E 0068           00112          CLRF    ACCA
004F 03E9           00113 TLOOP    INCFSZ  ACCA+1, F
0050 0A54           00114          GOTO    ENDCHK
0051 03E8           00115          INCFSZ  ACCA, F
0052 0A54           00116          GOTO    ENDCHK
0053 0A56           00117          GOTO    END_M
0054 0701           00118 ENDCHK   BTFSS   1,0            ;CHECK FOR TMR0 TRIP
0055 0A4F           00119          GOTO    TLOOP
0056 0201           00120 END_M    MOVF    1,W
0057 0800           00121          RETLW   0
```


© 1997 Microchip Technology Inc.

```
                    00122
0058 0C03           00123 OHMS   MOVLW   B'00000011'   ;SET RA0 AND RA1 HIGH (ON WHEN ACTIVATED)
0059 0025           00124        MOVWF   5
005A 0C28           00125        MOVLW   B'00101000'   ;SELECT POSITIVE EDGE FOR TMR0
005B 0002           00126        OPTION
                    00127
005C 0943           00128 CAL    CALL    DSCHRG        ;DISCHARGE CAPACITOR
005D 0C0E           00129        MOVLW   B'00001110'   ;ACTIVATE RA0
005E 0005           00130        TRIS    5
005F 094C           00131        CALL    M_TIME        ;MEASURE TIME
0060 0209           00132        MOVF    ACCA+1,W
0061 0033           00133        MOVWF   TCAL+1        ;STORE LSB
0062 0208           00134        MOVF    ACCA,W
0063 0032           00135        MOVWF   TCAL          ;STORE MSB
                    00136
0064 0943           00137 MEAS   CALL    DSCHRG        ;DISCHARGE CAPACITOR
0065 0C0D           00138        MOVLW   B'00001101'   ;ACTIVATE RA1
0066 0005           00139        TRIS    5
0067 094C           00140        CALL    M_TIME        ;MEASURE TIME
                    00141
0068 0C3C           00142        MOVLW   RCALLS        ;CALIBRATION LSB VALUE
0069 002B           00143        MOVWF   ACCB+1
006A 0C2F           00144        MOVLW   RCALMS        ;CALIBRATION MSB VALUE
006B 002A           00145        MOVWF   ACCB
                    00146
006C 0908           00147        CALL    MPY           ;MULTIPLY ACCA(MEAS) * ACCB(RCAL)
006D 0213           00148        MOVF    TCAL+1,W
006E 0029           00149        MOVWF   ACCA+1
006F 0212           00150        MOVF    TCAL,W
0070 0028           00151        MOVWF   ACCA
                    00152
0071 0923           00153        CALL    DIV           ;DIVIDE ACCB(MEAS * R) BY ACCA(TCAL)
                    00154
0072 0A58           00155        GOTO OHMS
                    00156
                    00157        END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXX-------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   116
Program Memory Words Free:   396

Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

NOTES:

© 1997 Microchip Technology Inc.

Analog to Digital Conversion Using a PIC16C54

Author: *Doug Cox*
Microchip Technology Inc.

INTRODUCTION

This application note describes a method for implementing analog to digital (A/D) conversion on the PIC16C5X series of microcontrollers. The converter requires only five external components and is software and hardware configurable for conversion resolutions from 6-bits up to 10-bits and conversion times of 250µs or longer. The method is usable for both voltage and current conversion and uses a software calibration technique that compensates for time and temperature drift as well as component errors. PIC16C5X microcontrollers are ideal for simple analog applications because:

- Very low cost
- Few external components required
- Fully programmable. PIC16C5X microcontrollers are offered as One-Time-Programmable (OTP) EPROM devices.
- Available off the shelf from distributors
- Calibration in software for improved measurement accuracy
- Power savings using Sleep mode
- Output pins have large, current source/sink capability to drive LED's directly

THEORY OF OPERATION

The application uses a capacitive charging circuit (Figure 1) to convert the input voltage to time, which can be easily measured using a microcontroller. First, the reference voltage is applied to the input voltage to the current converter (U1). The equivalent circuit is shown in Figure 2. This circuit provides a linearly variable current as a function of input voltage. The logarithmic characteristic that would occur if the input voltage was applied directly to an RC is not present. The capacitor C is charged up until the threshold on the chip input trips. This generates a software calibration value that is used to calibrate out most circuit errors, including; inaccuracies in the resistor and capacitor, changes in the input threshold voltage, and temperature variations. After the software calibration value is measured, the capacitor is discharged (Figure 3), and the input voltage is connected to VIN. The time to the trip threshold is measured for the input voltage and compared to the calibration value to determine the actual input voltage.

FIGURE 1: VOLTMETER A/D CONVERTER

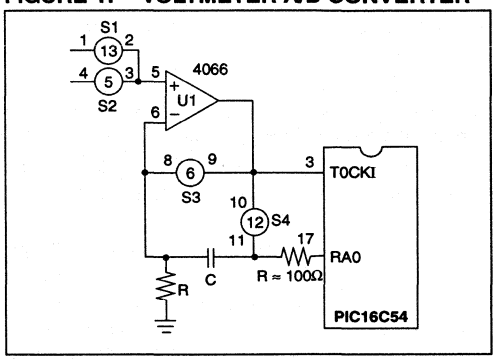


FIGURE 2: VOLTMETER MEASUREMENT CYCLE

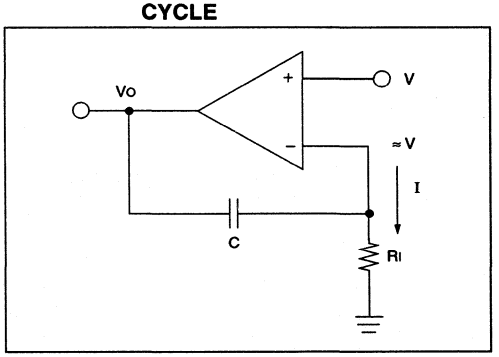
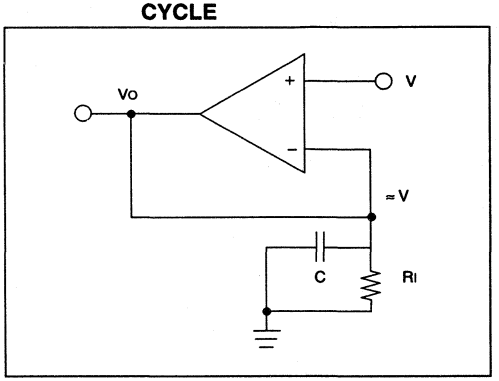


FIGURE 3: VOLTMETER DISCHARGE CYCLE



© 1997 Microchip Technology Inc.

CIRCUIT CONFIGURATION

The values of R and C are selected based upon the number of bits of resolution required.

$$RC = (V_I \bullet T)/V_T$$

Where:

V_I = Lowest voltage to be measured (at least ten LSb's)

T = Time to do the number of bits of resolution desired

V_T = Threshold voltage of the PIC16C5X input being used

Actual value for RC should be slightly smaller than calculated to ensure that the PIC16C5X does not overcount during the measurement.

For example use a 3V input and 8-bit resolution with a 8 MHz clock and 6 instruction cycles per count:

$V_I = 100\ mV$
$T = 256 \bullet 1/8\ MHz \bullet 4\ clocks/cycle \bullet 6\ cycles = 768\ \mu s$
$V_T = 3.0V\ (est)$

For input voltages greater than 3V a resistor divider network should be used to keep the maximum voltage on V_{IN} to less than 3V. For best performance the reference voltage should be between 2V and 3V.

The circuit can also be used as a current mode A/D converter. In this case the input voltage to current converter is not needed and the reference current and input current are both routed via analog switches directly into the capacitor.

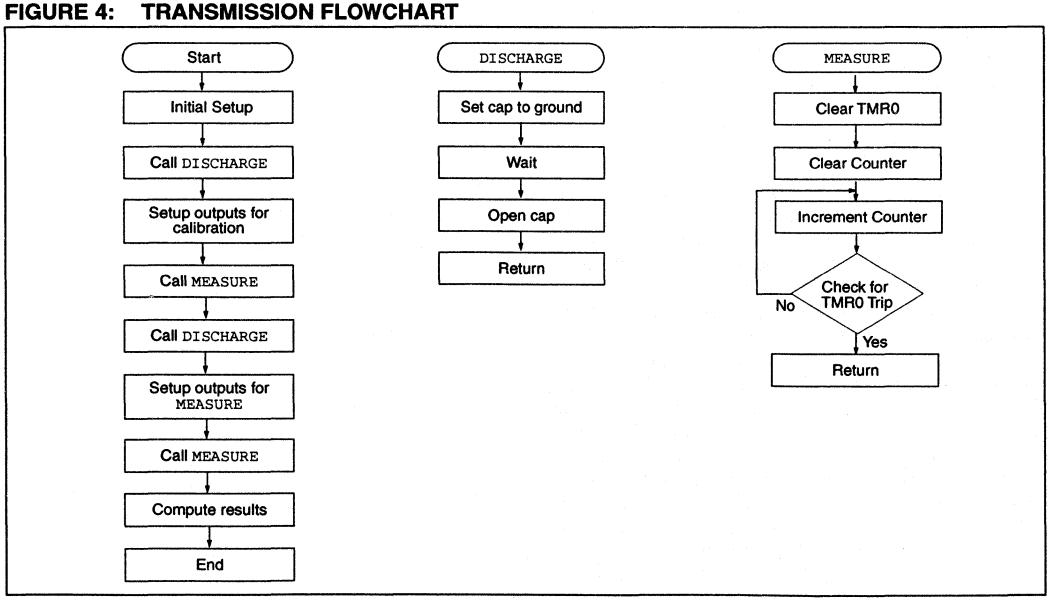
CIRCUIT PERFORMANCE

The calibration cycle removes all first order errors (offset, gain, R and C inaccuracy, power supply voltage and temperature) except the reference voltage drift. Any change in the reference voltage, including noise, between the calibration cycle and the measurement cycle may result in measurement errors. Other error sources may be analog switch leakage, resistor and capacitor non-linearities, input threshold uncertainty and time measurement uncertainty (± one instruction cycle time). Measured performance shows the converter to be accurate within ±1% of full scale.

Example

Assembly code implementing the circuit of Figure 1 is listed in Appendix A. This code measures the time up to 16-bits and calculates the results using 16-bit multiply and divide subroutines. In actual applications, if measurement accuracy permits, it may be advantageous to use 8-bits. The math code can be substantially reduced and the measure time is reduced by the simpler code and shorter count.

FIGURE 4: TRANSMISSION FLOWCHART



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: VOLTMETER/AD CONVERTER PROGRAM

```
MPASM 01.40 Released          VOLTMETR.ASM   1-16-1997  12:35:25         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         TITLE  'VOLTMETER/AD CONVERTER PROGRAM REV 3-29-90'
                     00002         LIST   P=16C54
                     00003 ;
                     00004 ;*****************************************************************
                     00005 ;
                     00006 ;       Program:          VOLTMETR.ASM
                     00007 ;       Revision Date:
                     00008 ;                       1-13-97      Compatibility with MPASMWIN 1.40
                     00009 ;
                     00010 ;*****************************************************************
                     00011 ;
  00000008           00012 ACCA     EQU     8
  0000000A           00013 ACCB     EQU     0A
  0000000C           00014 ACCC     EQU     0C
  0000000E           00015 ACCD     EQU     0E
  00000010           00016 ACCE     EQU     10
  00000012           00017 TMEAS    EQU     12
  00000014           00018 TEMP     EQU     14
                     00019 ;
  00000001           00020 F        EQU     1
                     00021 ;
  00000060           00022 VCALMS   EQU     60           ;VCAL MSB VALUE IN HEX
  000000A4           00023 VCALLS   EQU     0A4          ;VCAL LSB VALUE IN HEX
                     00024
01FF                 00025         ORG 1FF
01FF 0A58            00026         GOTO    VOLTS        ;PROGRAM CODE
0000                 00027         ORG     0            ;SUBROUTINES
                     00028
0000 0209            00029 MADD     MOVF    ACCA+1,W
0001 01EB            00030          ADDWF   ACCB+1, F    ;ADD LSB
0002 0603            00031          BTFSC   3,0          ;ADD IN CARRY
0003 02AA            00032          INCF    ACCB, F
0004 0208            00033          MOVF    ACCA,W
0005 01EA            00034          ADDWF   ACCB, F      ;ADD MSB
0006 0800            00035          RETLW   0
0007 0000            00036          NOP
                     00037
0008 0915            00038 MPY      CALL    SETUP        ;RESULTS IN B(16 MSB'S) AND C(16 LSB'S)
0009 032E            00039 MLOOP    RRF     ACCD, F      ;ROTATE D RIGHT
000A 032F            00040          RRF     ACCD+1, F
000B 0603            00041          SKPNC                ;NEED TO ADD?
000C 0900            00042          CALL    MADD
000D 032A            00043          RRF     ACCB, F
000E 032B            00044          RRF     ACCB+1, F
000F 032C            00045          RRF     ACCC, F
0010 032D            00046          RRF     ACCC+1, F
0011 02F4            00047          DECFSZ  TEMP, F      ;LOOP UNTIL ALL BITS CHECKED
0012 0A09            00048          GOTO    MLOOP
0013 0800            00049          RETLW   0
                     00050
0014 0000            00051          NOP
0015 0C10            00052 SETUP    MOVLW   10
0016 0034            00053          MOVWF   TEMP
0017 020A            00054          MOVF    ACCB,W       ;MOVE B TO D
```

```
0018 002E           00055           MOVWF   ACCD
0019 020B           00056           MOVF    ACCB+1,W
001A 002F           00057           MOVWF   ACCD+1
001B 020C           00058           MOVF    ACCC,W
001C 0030           00059           MOVWF   ACCE
001D 020D           00060           MOVF    ACCC+1,W
001E 0031           00061           MOVWF   ACCE+1
001F 006A           00062           CLRF    ACCB
0020 006B           00063           CLRF    ACCB+1
0021 0800           00064           RETLW   0
                    00065
0022 0000           00066           NOP
0023 0915           00067 DIV       CALL    SETUP
0024 0C20           00068           MOVLW   20
0025 0034           00069           MOVWF   TEMP
0026 006C           00070           CLRF    ACCC
0027 006D           00071           CLRF    ACCC+1
0028 0403           00072 DLOOP     CLRC
0029 0371           00073           RLF     ACCE+1, F
002A 0370           00074           RLF     ACCE, F
002B 036F           00075           RLF     ACCD+1, F
002C 036E           00076           RLF     ACCD, F
002D 036D           00077           RLF     ACCC+1, F
002E 036C           00078           RLF     ACCC, F
002F 0208           00079           MOVF    ACCA,W
0030 008C           00080           SUBWF   ACCC,W          ;CHECK IF A>C
0031 0743           00081           SKPZ
0032 0A35           00082           GOTO    NOCHK
0033 0209           00083           MOVF    ACCA+1,W
0034 008D           00084           SUBWF   ACCC+1,W        ;IF MSB EQUAL THEN CHECK LSB
0035 0703           00085 NOCHK     SKPC                    ;CARRY SET IF C>A
0036 0A3E           00086           GOTO    NOGO
0037 0209           00087           MOVF    ACCA+1,W        ;C-A INTO C
0038 00AD           00088           SUBWF   ACCC+1, F
0039 0703           00089           BTFSS   3,0
003A 00EC           00090           DECF    ACCC, F
003B 0208           00091           MOVF    ACCA,W
003C 00AC           00092           SUBWF   ACCC, F
003D 0503           00093           SETC                    ;SHIFT A 1 INTO B (RESULT)
003E 036B           00094 NOGO      RLF     ACCB+1, F
003F 036A           00095           RLF     ACCB, F
0040 02F4           00096           DECFSZ  TEMP, F         ;LOOP UNTILL ALL BITS CHECKED
0041 0A28           00097           GOTO    DLOOP
0042 0800           00098           RETLW   0
                    00099
0043 0C0E           00100 DSCHRG    MOVLW   B'00001110'     ;DISCHARGE C (RA0 ON)
0044 0005           00101           TRIS    5
0045 0CFF           00102           MOVLW   0FF
0046 0034           00103           MOVWF   TEMP
0047 02F4           00104 LOOP      DECFSZ  TEMP, F         ;WAIT
0048 0A47           00105           GOTO    LOOP
0049 0C0F           00106           MOVLW   B'00001111'     ;ALL RA HIGH Z
004A 0005           00107           TRIS    5
004B 0800           00108           RETLW   0
                    00109
004C 0061           00110 M_TIME    CLRF    1               ;CLEAR TMR0 REGISTER
004D 0069           00111           CLRF    ACCA+1          ;CLEAR 16 BIT COUNTER
004E 0068           00112           CLRF    ACCA
004F 03E9           00113 TLOOP     INCFSZ  ACCA+1, F
0050 0A54           00114           GOTO    ENDCHK
0051 03E8           00115           INCFSZ  ACCA, F
0052 0A54           00116           GOTO    ENDCHK
0053 0A56           00117           GOTO    END_M
0054 0701           00118 ENDCHK    BTFSS   1,0             ;CHECK FOR TMR0 TRIP
0055 0A4F           00119           GOTO    TLOOP
0056 0201           00120 END_M     MOVF    1,W
```


© 1997 Microchip Technology Inc.

```
0057 0800           00121          RETLW   0
                    00122
0058 0C06           00123 VOLTS    MOVLW   B'00000110'  ;SET S2 AND S3 HIGH(ON WHEN ACTIVATED)
0059 0026           00124          MOVWF   6
005A 0CF0           00125          MOVLW   B'11110000'  ;ACTIVATE SWITCHES S1-S4
005B 0006           00126          TRIS    6
005C 0C28           00127          MOVLW   B'00101000'  ;SELECT POSITIVE EDGE FOR TMR0
005D 0002           00128          OPTION
005E 0C00           00129          MOVLW   B'00000000'
005F 0025           00130          MOVWF   5            ;SET RA0 LOW (ON WHEN ACTIVATED)
                    00131
0060 0943           00132 MEAS     CALL    DSCHRG       ;CHARGE CAPACITOR TO VIN
0061 0C0A           00133          MOVLW   B'00001010'  ;S2 AND S4 ON
0062 0026           00134          MOVWF   6
0063 094C           00135          CALL    M_TIME       ;MEASURE TIME
0064 0209           00136          MOVF    ACCA+1,W
0065 0033           00137          MOVWF   TMEAS+1      ;STORE LSB
0066 0208           00138          MOVF    ACCA,W
0067 0032           00139          MOVWF   TMEAS        ;STORE MSB
                    00140
0068 0C05           00141 CAL      MOVLW   B'00000101'  ;S1 AND S3 ON
0069 0026           00142          MOVWF   6
006A 0943           00143          CALL    DSCHRG       ;CHARGE CAPACITOR TO VREF
006B 0C09           00144          MOVLW   B'00001001'  ;S1 AND S4 ON
006C 0026           00145          MOVWF   6
006D 094C           00146          CALL    M_TIME       ;MEASURE TIME
                    00147
006E 0CA4           00148          MOVLW   VCALLS
006F 002B           00149          MOVWF   ACCB+1
0070 0C60           00150          MOVLW   VCALMS
0071 002A           00151          MOVWF   ACCB
                    00152
0072 0908           00153          CALL    MPY          ;MULTIPLY ACCA(TCAL) * ACCB(VREF)
0073 0213           00154          MOVF    TMEAS+1,W
0074 0029           00155          MOVWF   ACCA+1
0075 0212           00156          MOVF    TMEAS,W
0076 0028           00157          MOVWF   ACCA
                    00158
0077 0923           00159          CALL    DIV          ;DIVIDE ACCB(TCAL * V) BY ACCA(TMEAS)
                    00160
0078 0A58           00161          GOTO VOLTS
                    00162
                    00163          END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXX-------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   122
Program Memory Words Free:   390


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Software Interrupt Techniques

INTRODUCTION

This application note describes a unique method for implementing interrupts in software on the PIC16C5X series of microcontrollers. This method takes advantage of the PIC16C5X's architecture which allows changing the program counter under software control. Up to eight interrupt lines are possible, but the practical limit for simple code generation is six interrupts, or 64 possible input conditions. The interrupt detection time is under software control and standard I/O pins are used as the interrupt lines.

THEORY OF OPERATION

SOFTWARE POLLING OF I/O LINES TO REPLACE HARDWARE INTERRUPTS

The interrupt conditions are determined by detecting changes on the I/O lines that have been selected to be the interrupt lines. These changes are used to create a jump table that allows a different program response to each interrupt condition. The interrupt response time is under software control and can be as short as ten to twenty microseconds, depending on main program and interrupt subroutine program length.

CREATING THE INTERRUPT SUBROUTINE JUMP TABLE

Each I/O condition may have its own unique subroutine to respond to changes on the interrupt lines. Direct access to these routines is achieved by using the PIC16C5X's ability to change the program counter under software control. Here is an example of how two I/O lines may be polled:

```
MOVF    CONDTN,W ;LOAD I/O CONDITION INTO W
                 ;REGISTER
ANDLW   3        ;MASK OFF TOP 6 BITS
ADDWF   2,1      ;ADD INPUT TO PROGRAM COUNTER
                 ;TO CREATE JUMP TABLE
GOTO    MAIN     ;FOR NO CHANGE GOTO MAIN
                 ;PROGRAM
GOTO    INT1     ;FOR CHANGE IN BIT 0 GOTO INT1
GOTO    INT2     ;FOR CHANGE IN BIT 1 GOTO INT2
GOTO    INT3     ;FOR BOTH CHANGES GOTO INT3
```

The changes to the I/O lines have been used to create a two bit number that is added to the program counter. The GOTO that is executed depends on the new program counter address.

CREATING CONSTANT TIME POLLING

In most applications requiring interrupts, it is important to poll the interrupt lines at fixed time intervals, usually only a few microseconds in length. Two techniques may be used on the PIC16C5X to achieve this timing. One divides the main program into multiple sections and the other implements an elapsed time counter (Figure 1). Both of these techniques use the same program jump table concept that was described earlier. In the first technique, the main program is divided into several sections based on the desired I/O polling time. When MAIN is called a branch register is added to the program counter. This determines which section of MAIN code should be executed next. At the end of execution the branch register is decremented so the next section of code will be executed after the next polling. If the branch register is zero, then the number of sections of main code is added into it to start the main program over again.

In the second technique, an elapsed time counter can be implemented using the Timer0 counter. At the beginning of I/O polling the TMR0 register is cleared. It then starts counting the instruction cycles. Then after the main program subsection has been executed, the TMR0 register value is subtracted from the desired polling time. This determines how many instructions need to be executed before the next polling. A jump table is then created to execute these instructions before the next polling. An example is shown below. This example assumes from zero to 15 additional instruction cycles are needed. Actual numbers need to be computed for each individual application.

```
MOVLW   POLL   ;POLL=DESIRED POLL CYCLES-15
SUBWF   TMR0,W;DETERMINE HOW MUCH TIME TO WAIT
ADDWF   2,1    ;ADD WAIT TIME TO PROGRAM COUNTER
NOP            ;15 ADDITIONAL INSTRUCTION CYCLES
:
:              ;TOTAL OF 15 NOP'S
NOP            ;1 ADDITIONAL INSTRUCTION CYCLE
GOTO    START  ;0 ADDITIONAL INSTRUCTION CYCLES
```

© 1997 Microchip Technology Inc.

For example, if the desired instruction time is 50 cycles and the subsection we just executed has a consumed a total of 40 instruction cycles (including all overhead cycles) the value of:

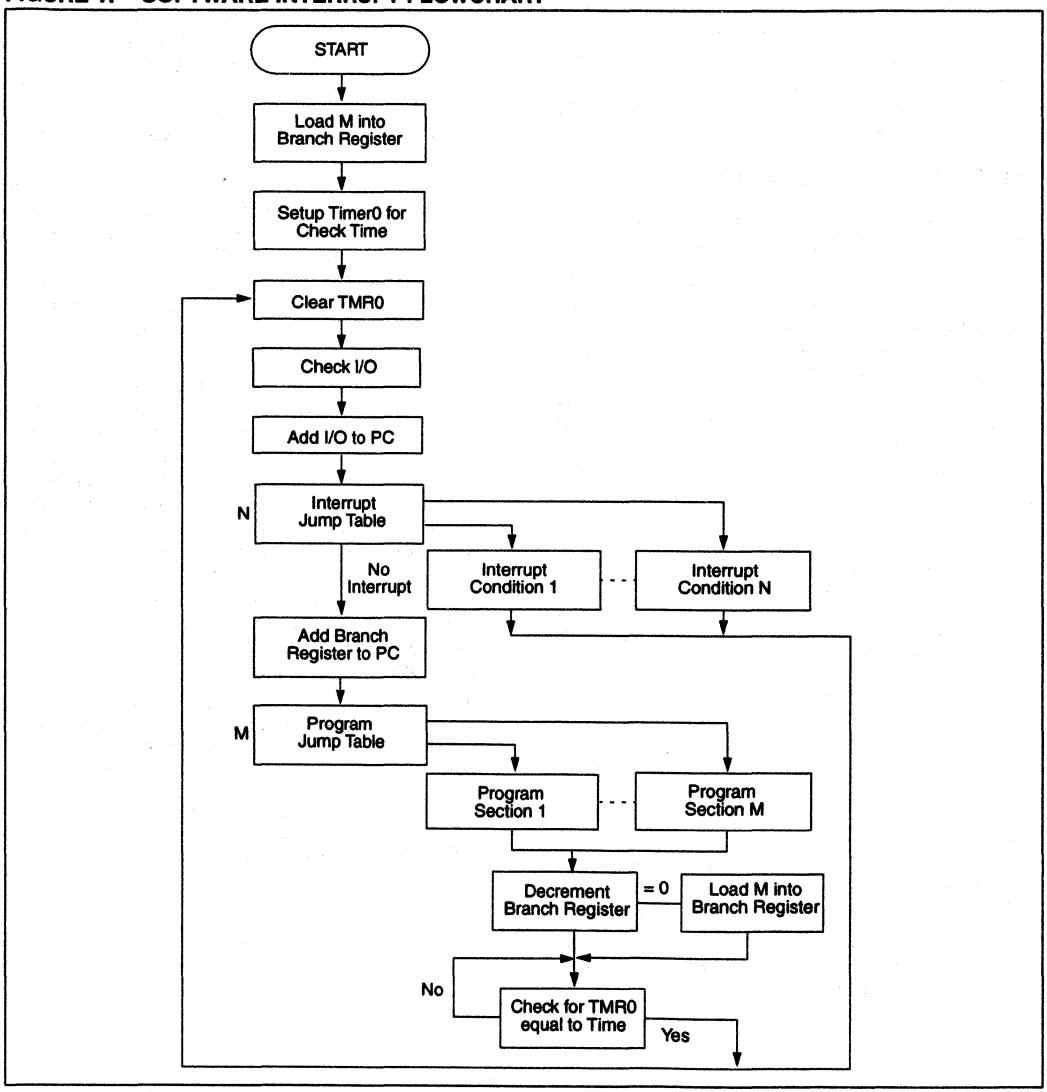
$$TMR0(40) - POLL(50 - 15(= 35)) = 5$$

will be added to the program counter. The program will then jump to the sixth NOP. That NOP, plus the nine following it, will be executed for a total of ten more instruction cycles. Note that the final GOTO has two instruction cycles and these must be included in the program overhead.

Example

The following example (Figure 1 and Appendix A) is the core program for the software interrupt technique described previously. This program assumes four interrupt conditions, four main program sections, and eight additional elapsed time instructions.

FIGURE 1: SOFTWARE INTERRUPT FLOWCHART



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40 Released          INTRUPT.ASM   1-16-1997  12:37:20         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         TITLE 'SOFTWARE INTERRUPT PROGRAM REV 3-29-90'
                     00002         LIST    P=16C54
                     00003
                     00004 ;
                     00005 ;*****************************************************************
                     00006 ;
                     00007 ;    SOFTWARE INTERRUPT APPLICATIONS
                     00008 ;    BRANCH IS MAIN PROGRAM REGISTER
                     00009 ;
                     00010 ;       Program:          OHMETER.ASM
                     00011 ;       Revision Date:
                     00012 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00013 ;
                     00014 ;*****************************************************************
                     00015 ;
  00000008           00016 BRANCH EQU     8
  00000009           00017 CNDTN  EQU     9
  0000000A           00018 IO     EQU     0A
  0000000B           00019 TEMP   EQU     0B
                     00020
0000 0069            00021 SETUP  CLRF    CNDTN
0001 0C04            00022        MOVLW   4
0002 0028            00023        MOVWF   BRANCH    ;FOUR MAIN PROGRAM SECTIONS
0003 0C08            00024        MOVLW   8
0004 0002            00025        OPTION            ;SET TMR0 TO ONE COUNT PER INSTRUCTION CYCLE
                     00026
0005 0061            00027 START  CLRF    1         ;CLEAR TMR0 REGISTER
0006 0206            00028        MOVF    6,W       ;READ I/O
0007 002A            00029        MOVWF   IO
0008 0109            00030        IORWF   CNDTN,W   ;THIS SECTION OF CODE CALCULATES THE
0009 002B            00031        MOVWF   TEMP      ;JUMP TABLE. ANY INPUT THAT CHANGES FROM
000A 0209            00032        MOVF    CNDTN,W   ;A ZERO TO A ONE IS CONSIDERED AN INTERRUPT.
000B 00AB            00033        SUBWF   TEMP,1    ;THE EQUATION IS:
000C 020A            00034        MOVF    IO,W      ;  (IO + CNDTN) - CNDTN = INTERRUPT
000D 0029            00035        MOVWF   CNDTN     ;WHERE IO IS CURRENT INPUT AND
000E 020B            00036        MOVF    TEMP,W    ;CNDTN IS PREVIOUS INPUT.
000F 0E03            00037        ANDLW   3         ;MASK OFF TOP 6 BITS
0010 01E2            00038        ADDWF   2,1       ;ADD INPUT TO PC TO CREATE JUMP TABLE
0011 0A1B            00039        GOTO    MAIN      ;FOR INPUT=00
0012 0A15            00040        GOTO    INT1      ;FOR INPUT=01
0013 0A17            00041        GOTO    INT2      ;FOR INPUT=10
0014 0A19            00042        GOTO    INT3      ;FOR INPUT=11
                     00043
0015 0000            00044 INT1   NOP               ;INTERRUPT LINE 1 CODE
0016 0A05            00045        GOTO    START
0017 0000            00046 INT2   NOP               ;INTERRUPT LINE 2 CODE
0018 0A05            00047        GOTO    START
0019 0000            00048 INT3   NOP               ;INTERRUPT LINES 1 AND 2 CODE
001A 0A05            00049        GOTO    START
                     00050
001B 0208            00051 MAIN   MOVF    BRANCH,W
001C 01E2            00052        ADDWF   2,1       ;ADD BRANCH TO PC TO CREATE JUMP TABLE
001D 0000            00053        NOP
001E 0A28            00054        GOTO    MAIN4     ;JUMP TABLE, LAST FIRST ON DECREMENT TABLE
```

```
001F 0A26           00055         GOTO    MAIN3
0020 0A24           00056         GOTO    MAIN2
0021 0A22           00057         GOTO    MAIN1
                    00058
0022 0000           00059 MAIN1   NOP             ;MAIN PROGRAM CODE BANK ONE
0023 0A2A           00060         GOTO    BRNCHK
0024 0000           00061 MAIN2   NOP             ;MAIN PROGRAM CODE SECTION TWO
0025 0A2A           00062         GOTO    BRNCHK
0026 0000           00063 MAIN3   NOP             ;MAIN PROGRAM CODE SECTION THREE
0027 0A2A           00064         GOTO    BRNCHK
0028 0000           00065 MAIN4   NOP             ;MAIN PROGRAM CODE SECTION FOUR
0029 0A2A           00066         GOTO    BRNCHK
                    00067
002A 02E8           00068 BRNCHK  DECFSZ  BRANCH,1 ;DECREMENT BRANCH REGISTER AND CHECK FOR ZERO
002B 0A2E           00069         GOTO    TIMCHK
002C 0C04           00070         MOVLW   4
002D 0028           00071         MOVWF   BRANCH  ;RELOAD BRANCH WITH 4 AT END OF MAIN
                    00072
002E 0C29           00073 TIMCHK  MOVLW   D'41'    ;CHECK TO SEE IF TMR0 HAS REACHED 50(50-7)
002F 0081           00074         SUBWF   1,W     ;DETERMINE WAIT TIME
0030 01E2           00075         ADDWF   2,1     ;ADD WAIT TIME TO PC
0031 0000           00076         NOP
0032 0000           00077         NOP
0033 0000           00078         NOP
0034 0000           00079         NOP
0035 0000           00080         NOP
0036 0000           00081         NOP
0037 0000           00082         NOP
0038 0A05           00083         GOTO    START
                    00084         END

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXX-------

All other memory blocks unused.

Program Memory Words Used:    57
Program Memory Words Free:   455


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN519

Implementing a Simple Serial Mouse Controller

INTRODUCTION

The mouse is becoming increasingly popular as a standard pointing data entry device. There is no doubt that the demand for the mouse is increasing. Various kinds of mice can be found in the market, including optical mice, opto-mechanical mice, and their close relative, trackballs. The mouse interfaces to the host via a dedicated interface card or an RS-232 port. Their mechanisms are very similar. The major electrical components of a mouse are:

- Microcontroller
- Photo-transistors
- Infrared emitting diode
- Voltage conversion circuit

The intelligence of the mouse is provided by the microcontroller, therefore the features and performance of a mouse is greatly related to the microcontroller used.

This application note describes the implementation of a serial mouse using the PIC16C54. The PIC16C54 is a high speed 8-bit CMOS microcontroller offered by Microchip Technology Inc. It is an ideal candidate for a mouse controller.

THEORY OF OPERATION

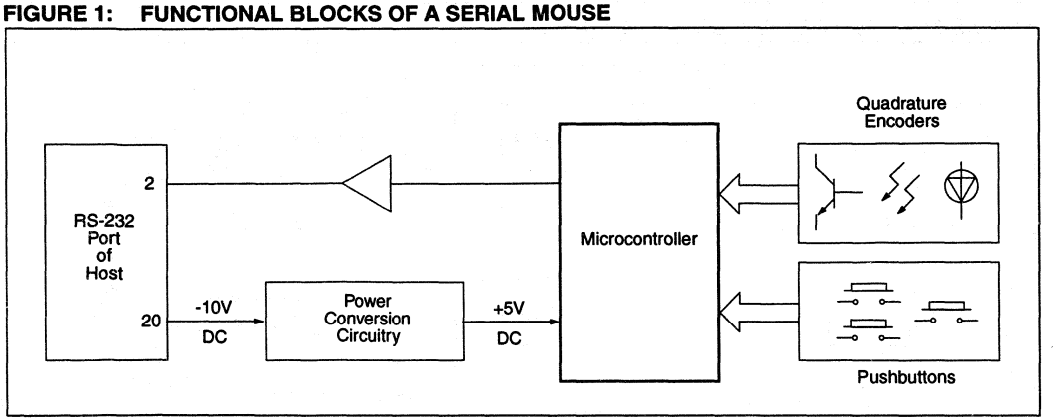
A mouse can be divided into several functional blocks:

- Microcontroller
- Button detection
- Motion detection
- RS-232 signal generation
- 5V DC power supply unit

A typical functional block diagram is shown in Figure 1.

In Figure 2, three pushbuttons are connected to the input ports of the PIC16C54. When a switch opening or closure is detected, a message is formatted and sent to the host. The X and Y movements are measured by counting the pulses generated by the photo-couplers. In the case of an opto-mechanical mouse, the infrared light emitted by the infrared diode is blocked by the rotating wheel, so that the pulses are generated on the photo-transistor side. In case of an optical mouse, the infrared light emitted by the infrared diode is reflected off the reflective pad patterned with vertical and horizontal grid lines. It is then received by the photo-transistor in the mouse. When any X or Y movement is detected, a message is formatted and sent to the host.

FIGURE 1: FUNCTIONAL BLOCKS OF A SERIAL MOUSE



© 1997 Microchip Technology Inc.

The Microsoft® Mouse System and the Mouse Systems® device both use serial input techniques. The Mouse System protocol format contains five bytes of data. One byte describes the status of three push buttons, two bytes for the relative X movements and two bytes for the relative Y movements. The Microsoft protocol format contains three bytes of data describing the status of two push buttons and the relative X and Y movements. The details of these protocols are given in Table 1.

Three lines are connected to the host via the RS-232 port:

- Signal Ground
- Received Data
- Request to Send

"Received Data" carries the message sent by the mouse. While "Request to Send" provides a –10 VDC for voltage conversion circuitry. A voltage of +5 VDC is required for electronic components inside the mouse, however, +5 VDC is not part of an RS-232 port, so voltage conversion circuitry is required. This circuit is typically composed of a 555 timer, Zener diodes, and capacitors.

An example circuit is shown in Figure 3. Since the current supplied through the RS-232 port is limited to 10 mA, the mouse cannot be designed to consume more than 10 mA current unless an external power supply is provided. The PIC16C54, running at 4 MHz (1 µs instruction cycle) can provide a very high tracking speed. An 8 MHz version of PIC16C54 is also available if higher performance is desired.

FIGURE 2: PIC16C54 PIN ASSIGNMENTS

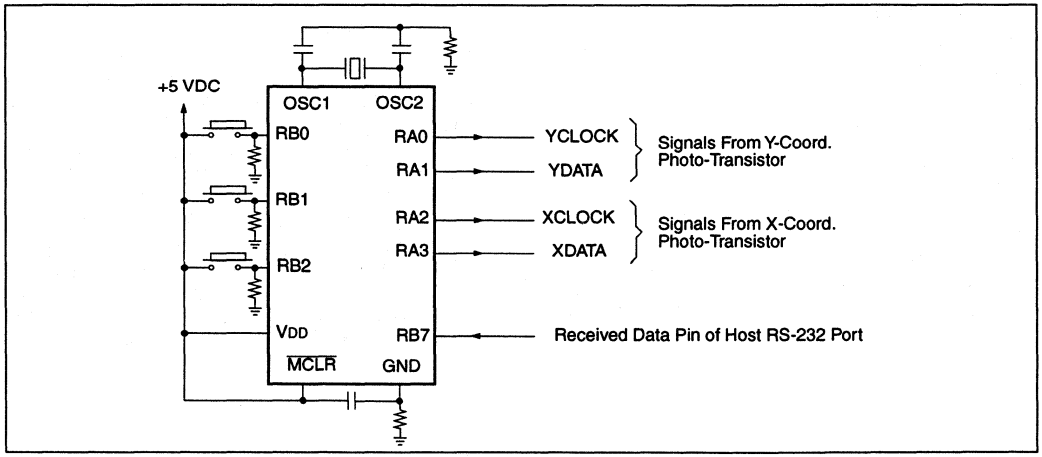
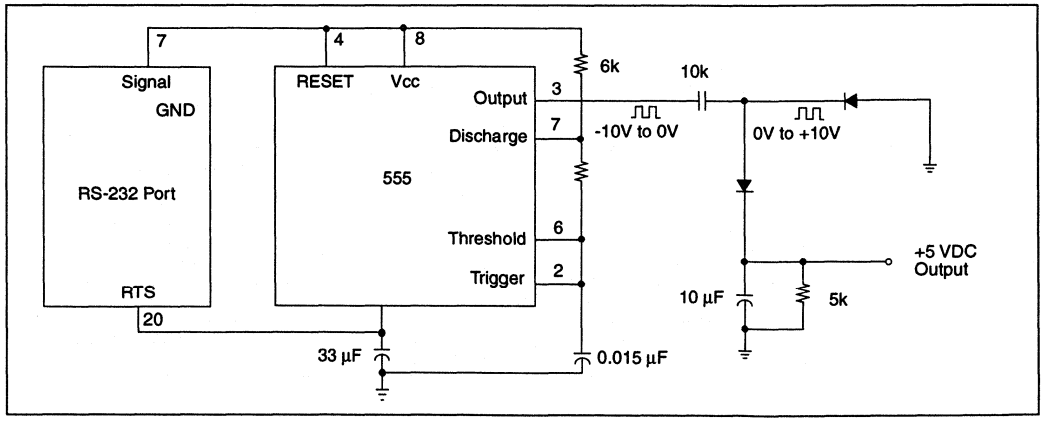


FIGURE 3: VOLTAGE CONVERSION CIRCUITRY



© 1997 Microchip Technology Inc.

ABOUT THE SOFTWARE

The major tasks performed by the software are button scanning, X and Y motion scanning, formatting and sending serial data to the host. These tasks need to be performed in parallel in order to gain better tracking speed. The pulses generated by the photo-couplers are counted while transmitting the serial signals to the RS-232 port. The number of pulses reflects the speed of the movement. The more pulses, the faster the movement.

The directions of movement are determined by the last states and the present states of the outputs of the photo-transistors. In Figure 4, XCLOCK and XDATA are outputs from the photo-transistors corresponding to the X-axis movement. XDATA is read when a rising or a falling edge of XCLOCK is detected. For right movement, XDATA is either LOW at the rising edge of XCLOCK or HIGH at the falling edge of XCLOCK. The up and down movement detections follow the same logic. In Table 1, X7:X0 are data for relative movement. If X is positive, it implies that the mouse is moving to the right. If X is negative, it implies a movement to the left. Similarly, if Y is positive, it indicates that the mouse is moving down and if Y is negative, it indicates that the mouse is moving up. The pulses generated by the photo-couplers are checked before every bit is sent. A bit takes 1/1200 second to send, if the distance between the grid lines is 1 mm, the tracking speed will be up to 1200 mm/second.

FIGURE 4: VOLTAGE CONVERSION CIRCUITRY

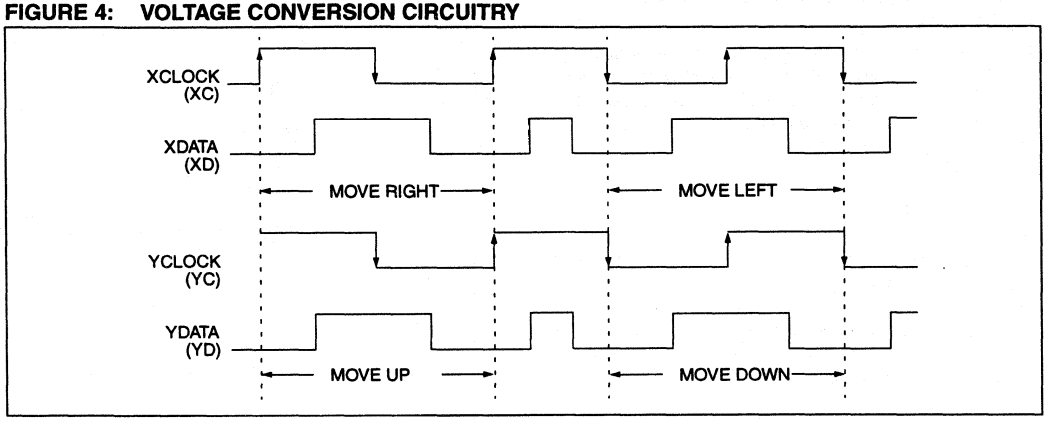


TABLE 1: MOUSE SYSTEM AND MICROSOFT PROTOCOLS

	Mouse System Format*								Microsoft Format*							
Bit Position	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 1	1	0	0	0	0	L	M	R	1	1	L	R	Y7	Y6	X7	X6
Byte 2	X7	X6	X5	X4	X3	X2	X1	X0	0	0	X5	X4	X3	X2	X1	X0
Byte 3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	0	0	Y5	Y4	Y3	Y2	Y1	Y0
Byte 4	X7	X6	X5	X4	X3	X2	X1	X0								
Byte 5	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0								

* L = Left Key Status
M = Middle Key Status
R = Right Key Status

1 = Pressed
0 = Released

X7-X0 = X-Axis Movement Data
Y7-Y0 = Y-Axis Movement Data

© 1997 Microchip Technology Inc.

The buttons are scanned after a message is sent and the time used to send the message is used as the debouncing time. The message is in an RS-232 format with 1200 baud, eight data bits, no parity, and two stop bits.

The flowcharts of the main program, subroutine BYTE and subroutine BIT are shown in Figure 5, Figure 6, and Figure 7. Figure 5 shows that the trigger flag is set when any change in button status or X/Y movement is detected. Subroutine BYTE is called in the main program five times to send five bytes of information. Subroutine BYTE controls the status of the "Received Data" (RD) pin. If Trigger Flag is clear, RD will always be HIGH. Hence, no message will be sent even when subroutine BYTE is called. Figure 7 shows that subroutine BIT counts the number of pulses from the outputs of the photo-transistors, determines the directions, and generates a 1/1200 second delay to get 1200 baud timing.

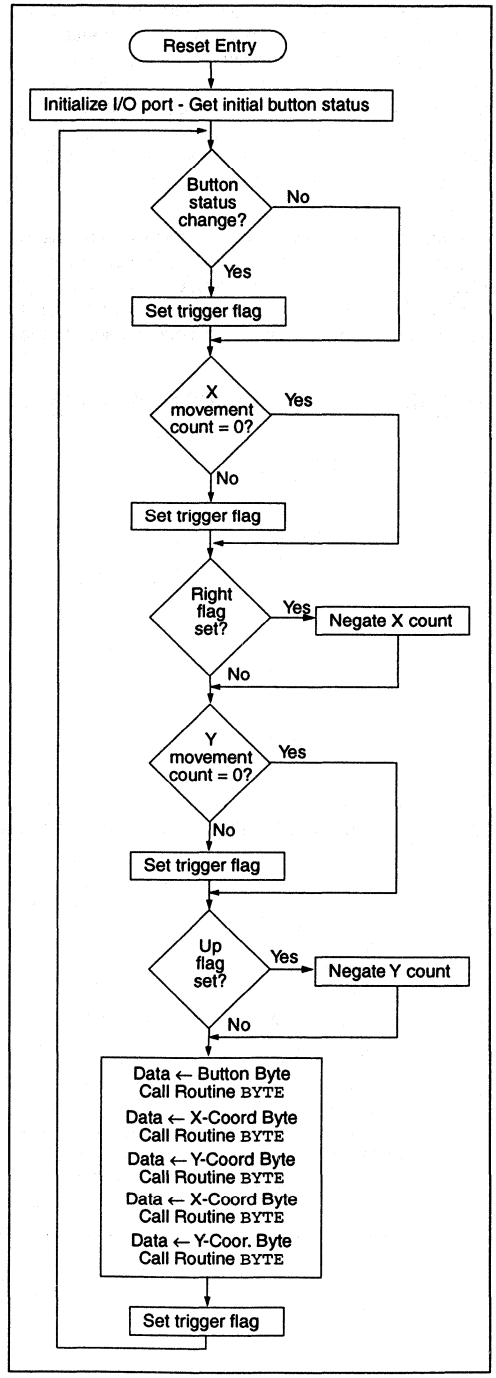
The mouse has been tested in Mouse System Mode and functions properly. The setup and software have been tested and function within the given guidlines. A listing of the source program is given in Appendix A.

SUMMARY

The PIC16C54 from Microchip Technology Inc. provides a very cost-effective, high performance mouse implementation. Its low power consumption (typically< 2 mA at 1 µs instruction cycle), small package (18-pin) and high reliability (on-chip watchdog timer to prevent software hang-ups) are some of the many reasons why the PIC16C54 is uniquely suitable for mouse applications.

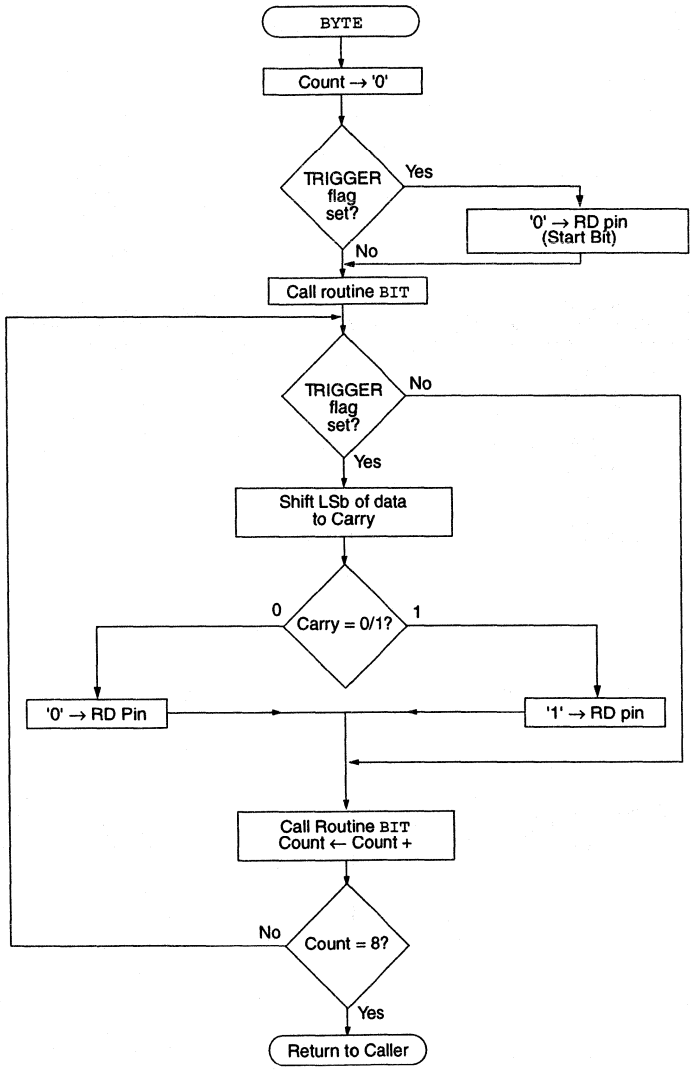
Note: This application note provides the user with a simple, fully functional serial mouse implementation. The user may use this as a starting point for a more comprehensive design.

FIGURE 5: FLOWCHART OF THE MAIN PROGRAM



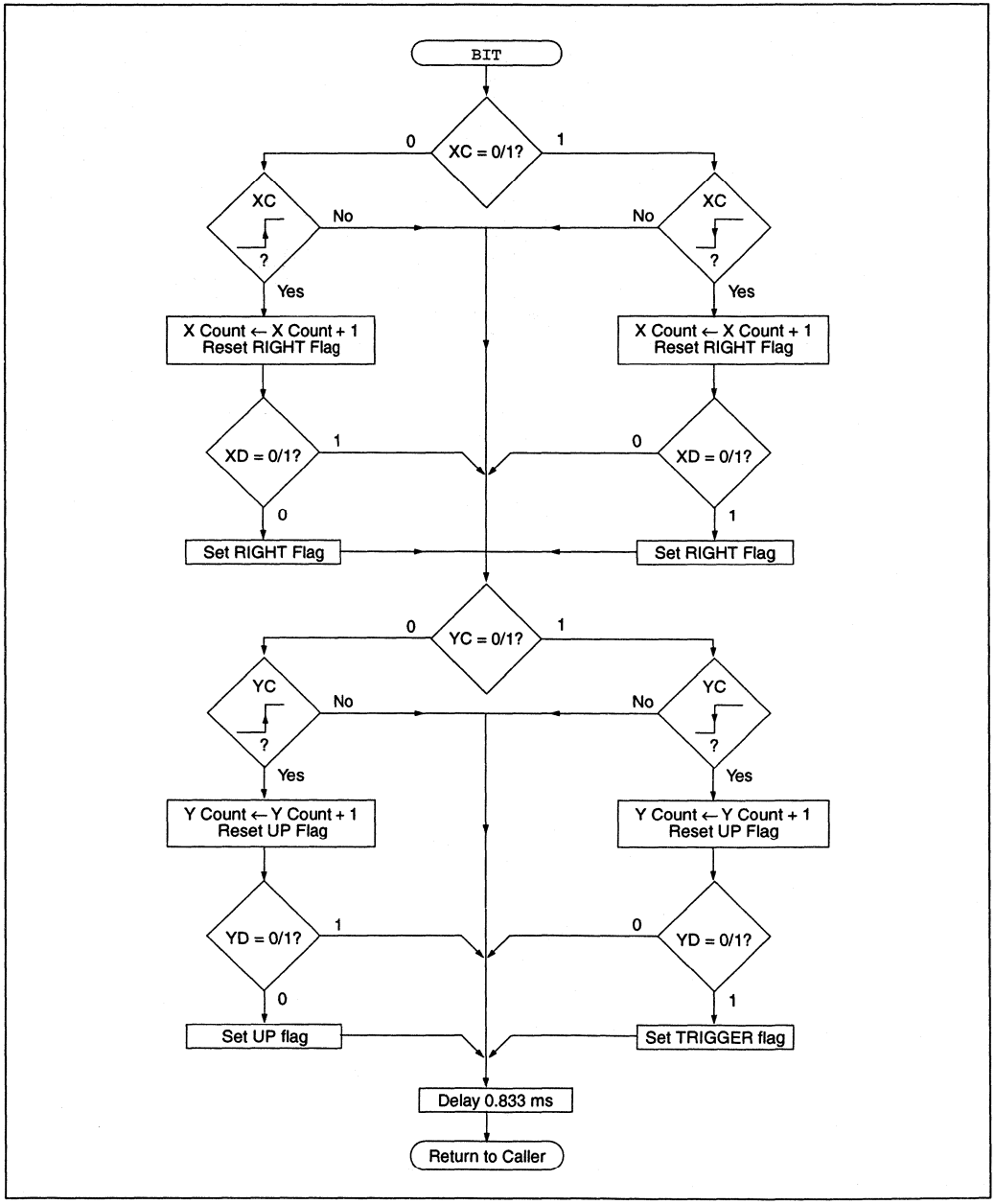
© 1997 Microchip Technology Inc.

FIGURE 6: FLOWCHART OF ROUTINE BYTE



© 1997 Microchip Technology Inc.

FIGURE 7: FLOWCHART OF ROUTINE BIT



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40 Released            MOUSE.ASM   1-16-1997  12:44:22         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001  TITLE " MOUSE "
                     00002  LIST   P=16C54,R=0
                     00003 ;
                     00004 ;
                     00005 ;*****************************************************************
                     00006 ;
                     00007 ;     MOUSE CONTROLLER
                     00008 ;     PIC MODE = PIC16C54XT   CLK=4.0MHZ
                     00009 ;
                     00010 ;        Program:          MOUSE.ASM
                     00011 ;        Revision Date:    25 APRIL, 1990
                     00012 ;                          1-13-97      Compatibility with MPASMWIN 1.40
                     00013 ;
                     00014 ;*****************************************************************
                     00015 ;
                     00016 ;-----------------------------------------
                     00017 ;        FILES ASSIGNMENT
                     00018 ;-----------------------------------------
                     00019 ;
00000003             00020 STATUS     EQU 3            ;STATUS REGISTER
00000005             00021 RA         EQU 5            ;I/O PORT A
00000006             00022 RB         EQU 6            ;I/O PORT B
00000008             00023 TIMER1     EQU 10           ;COUNTER FOR DELAY
0000000C             00024 CSTAT      EQU 14           ;CO-ORDINATE STATUS
0000000D             00025 BSTAT      EQU 15           ;BUTTON STATUS
0000000E             00026 DATA0      EQU 16           ;
0000000F             00027 DATA1      EQU 17           ;
00000010             00028 DATA2      EQU 20           ;5 BYTE RS232 DATA
00000011             00029 DATA3      EQU 21           ;
00000012             00030 DATA4      EQU 22           ;
00000013             00031 FLAGA      EQU 23           ;GENERAL PURPOSE FLAG
00000014             00032 XCOUNT     EQU 24           ;X-MOVEMENT COUNTER
00000015             00033 YCOUNT     EQU 25           ;Y-MOVEMENT COUNTER
00000016             00034 FLAGB      EQU 26           ;GENERAL PURPOSE FLAG
00000018             00035 COUNT      EQU 30           ;GENERAL PURPOSE COUNTER
00000019             00036 DATA_AREA  EQU 31           ;FOR TEMP. STORAGE
                     00037 ;
                     00038 ;-----------------------------------------
                     00039 ;        BIT ASSIGNMENT
                     00040 ;-----------------------------------------
                     00041 ;
00000000             00042 YC         EQU 0               ;Y-CLOCK PIN
00000001             00043 YD         EQU 1               ;Y-DATA PIN
00000001             00044 UP         EQU 1               ;MOVING UP FLAG
00000002             00045 XC         EQU 2               ;X-CLOCK PIN
00000003             00046 XD         EQU 3               ;X-DATA PIN
00000003             00047 RI         EQU 3               ;MOVING RIGHT FLAG
00000000             00048 BU1        EQU 0               ;BUTTON #1 PIN
00000002             00049 BU2        EQU 2               ;BUTTON #2 PIN
00000000             00050 CA         EQU 0               ;CARRY FLAG
00000007             00051 RD         EQU 7               ;RECEIVED DATA PIN TO RS232
00000002             00052 ZERO_AREA  EQU 2               ;ZERO FLAG
00000002             00053 TR         EQU 2               ;TIGGER FLAG
                     00054 ;
```

```
  00000001          00055 F          EQU 1
                    00056 ;
                    00057 ;=========================================
                    00058 ;       SUBROUTINES
                    00059 ;=========================================
                    00060 ;
                    00061 ;*****************************************
0000                00062  ORG 0
                    00063 ;*****************************************
                    00064 ;
                    00065 ;=========================================
                    00066 ; DELAY A BIT TIME AND CHECK XC & YC STATUS
                    00067 ;=========================================
0000                00068 BIT
0000 0745           00069  BTFSS   RA,XC            ;XC = 1 ?
0001 0A0A           00070  GOTO    BIT0
0002 064C           00071  BTFSC   CSTAT,XC         ;(XC=1)
0003 0A11           00072  GOTO    BITY            ;(XC ALWAYS = 1)
0004 02B4           00073  INCF    XCOUNT, F       ;(XC --|__)
0005 0476           00074  BCF     FLAGB,RI        ;DEFAULT LEFT
0006 0765           00075  BTFSS   RA,XD           ;LEFT / RIGHT ?
0007 0A11           00076  GOTO    BITY
0008 0576           00077  BSF     FLAGB,RI
0009 0A11           00078  GOTO    BITY
000A                00079 BIT0
000A 074C           00080  BTFSS   CSTAT,XC         ;(XC=0)
000B 0A11           00081  GOTO    BITY            ;(XC ALWAYS = 0)
000C 02B4           00082  INCF    XCOUNT, F       ;(XC __|--)
000D 0476           00083  BCF     FLAGB,RI        ;DEFAULT LEFT
000E 0665           00084  BTFSC   RA,XD           ;LEFT / RIGHT ?
000F 0A11           00085  GOTO    BITY
0010 0576           00086  BSF     FLAGB,RI
0011                00087 BITY
0011 0705           00088  BTFSS   RA,YC           ;YC = 1 ?
0012 0A1B           00089  GOTO    BITY0
0013 060C           00090  BTFSC   CSTAT,YC        ;(YC=1)
0014 0A22           00091  GOTO    BITDY           ;(YC ALWAYS = 1)
0015 02B5           00092  INCF    YCOUNT, F       ;(YC --|__)
0016 0436           00093  BCF     FLAGB,UP        ;DEFAULT DOWN
0017 0725           00094  BTFSS   RA,YD           ;DOWN / UP ?
0018 0A22           00095  GOTO    BITDY
0019 0536           00096  BSF     FLAGB,UP
001A 0A22           00097  GOTO    BITDY
001B                00098 BITY0
001B 070C           00099  BTFSS   CSTAT,YC        ;(YC=0)
001C 0A22           00100  GOTO    BITDY           ;(YC ALWAYS = 0)
001D 02B5           00101  INCF    YCOUNT, F       ;(YC __|--)
001E 0436           00102  BCF     FLAGB,UP        ;DEFAULT DOWN
001F 0625           00103  BTFSC   RA,YD           ;DOWN / UP ?
0020 0A22           00104  GOTO    BITDY
0021 0536           00105  BSF     FLAGB,UP
0022                00106  BITDY
0022 0205           00107  MOVF    RA,W            ;SAVE COOR. STATUS
0023 002C           00108  MOVWF   CSTAT
0024 0CC1           00109  MOVLW   193D            ;0.833 MS DELAY
0025 0028           00110  MOVWF   TIMER1
0026                00111 BITD0
0026 0000           00112  NOP
0027 02E8           00113  DECFSZ  TIMER1, F
0028 0A26           00114  GOTO    BITD0
0029 0800           00115  RETLW 0
                    00116 ;
                    00117 ;=================================================
                    00118 ;
                    00119 ;*************************************************
                    00120 ;*      SUBROUTINE TO SEND A BYTE                *
```


© 1997 Microchip Technology Inc.

```
                00121 ;*      AS RS232C FORMAT 8,N,1                  *
                00122 ;*********************************************
                00123 ;
002A            00124 BYTE
002A 0078       00125  CLRF    COUNT             ;RESET 8 BIT COUNT
002B 0753       00126  BTFSS   FLAGA,TR          ;ANY TRIGGER
002C 0A2E       00127  GOTO    BYTE0
002D 04E6       00128  BCF     RB,RD             ;LOW RD FOR START BIT
002E            00129 BYTE0
002E 0900       00130  CALL    BIT
002F            00131 BYTE1
002F 0753       00132  BTFSS   FLAGA,TR          ;ANY TRIGGER ?
0030 0A37       00133  GOTO BYTE3
0031 0339       00134  RRF     DATA_AREA, F      ;SHIFT DATA TO CARRY
0032 0703       00135  BTFSS   STATUS,CA         ;0 / 1 ?
0033 0A36       00136  GOTO    BYTE2
0034 05E6       00137  BSF     RB,RD             ;SEND A 1
0035 0A37       00138  GOTO    BYTE3
0036            00139 BYTE2
0036 04E6       00140  BCF     RB,RD             ;SEND A 0
0037            00141 BYTE3
0037 0900       00142  CALL    BIT
0038 02B8       00143  INCF    COUNT, F
0039 0778       00144  BTFSS   COUNT,3           ;COUNT = 8 ?
003A 0A2F       00145  GOTO    BYTE1
003B 0753       00146  BTFSS   FLAGA,TR          ;ANY TRIGGER ?
003C 0A42       00147  GOTO    BYTE4
003D 04E6       00148  BCF     RB,RD             ;SEND SENT BIT
003E 0900       00149  CALL    BIT
003F 05E6       00150  BSF     RB,RD
0040 0900       00151  CALL    BIT
0041 0A44       00152  GOTO    BYTE5
0042            00153 BYTE4
0042 0900       00154  CALL    BIT
0043 0900       00155  CALL    BIT
0044            00156 BYTE5
0044 0800       00157  RETLW   0
                00158 ;
                00159 ;=========================================
                00160 ;       RESET ENTRY
                00161 ;=========================================
                00162 ;
0045            00163 INIT
0045 0CC1       00164  MOVLW   B'11000001'       ;DISABLE WATCHDOG
0046 0002       00165  OPTION
0047 0C0F       00166  MOVLW   B'00001111'       ;INIT RB0~3 BE INPUTS
0048 0006       00167  TRIS    RB                ;RB4~7 BE OUTPUTS
0049 0CFF       00168  MOVLW   B'11111111'       ;INIT RA0~3 BE INPUTS
004A 0005       00169  TRIS    RA
004B 05E6       00170  BSF     RB,RD             ;HIGH RD PIN
004C 0246       00171  COMF    RB,W              ;GET INIT BUTTON INPUTS
004D 0E05       00172  ANDLW   B'00000101'
004E 0D80       00173  IORLW   B'10000000'
004F 002D       00174  MOVWF   BSTAT
0050 002E       00175  MOVWF   DATA0
0051 0205       00176  MOVF    RA,W
0052 002C       00177  MOVWF   CSTAT
0053 0073       00178  CLRF    FLAGA             ;CLEAR TR FLAG
0054 0074       00179  CLRF    XCOUNT            ;RESET XCOUNT & YCOUNT
0055 0075       00180  CLRF    YCOUNT
0056            00181 SCAN
0056 006F       00182  CLRF    DATA1             ;UPDATE X,Y MOVEMENT DATA
0057 0070       00183  CLRF    DATA2
0058 0071       00184  CLRF    DATA3
0059 0072       00185  CLRF    DATA4
005A 0214       00186  MOVF    XCOUNT,W          ;XCOUNT = 0 ?
```


© 1997 Microchip Technology Inc.

```
005B 0743          00187  BTFSS   STATUS,ZERO_AREA
005C 0A80          00188  GOTO    WRITX
005D               00189 SCANA
005D 0215          00190  MOVF    YCOUNT,W          ;YCOUNT = 0 ?
005E 0743          00191  BTFSS   STATUS,ZERO_AREA
005F 0A92          00192  GOTO    WRITY
0060               00193 SCANB
0060 0246          00194  COMF    RB,W              ;BUTTON STATUS CHANGE ?
0061 0E05          00195  ANDLW   B'00000101'
0062 0D80          00196  IORLW   B'10000000'
0063 00AD          00197  SUBWF   BSTAT, F
0064 0643          00198  BTFSC   STATUS,ZERO_AREA  ;IF CHANGE THEN TRIGGER
0065 0A6B          00199  GOTO    SCANC             ;(NO CHANGE)
0066 0553          00200  BSF     FLAGA,TR          ;(CHANGE) SET TRIGGER FLAG
0067 0246          00201  COMF    RB,W              ;FORMAT BUTTON STATUS DATA
0068 0E05          00202  ANDLW   B'00000101'
0069 0D80          00203  IORLW   B'10000000'
006A 002E          00204  MOVWF   DATA0
006B               00205 SCANC
006B 0246          00206  COMF    RB,W
006C 0E05          00207  ANDLW   B'00000101'
006D 0D80          00208  IORLW   B'10000000'
006E 002D          00209  MOVWF   BSTAT
006F 020E          00210  MOVF    DATA0,W           ;SEND DATA0,1,2,3,4 TO HOST
0070 0039          00211  MOVWF   DATA_AREA
0071 092A          00212  CALL    BYTE
0072 020F          00213  MOVF    DATA1,W
0073 0039          00214  MOVWF   DATA_AREA
0074 092A          00215  CALL    BYTE
0075 0210          00216  MOVF    DATA2,W
0076 0039          00217  MOVWF   DATA_AREA
0077 092A          00218  CALL    BYTE
0078 0211          00219  MOVF    DATA3,W
0079 0039          00220  MOVWF   DATA_AREA
007A 092A          00221  CALL    BYTE
007B 0212          00222  MOVF    DATA4,W
007C 0039          00223  MOVWF   DATA_AREA
007D 092A          00224  CALL    BYTE
007E 0453          00225  BCF     FLAGA,TR          ;CLEAR TRIGGER FLAG
007F 0A56          00226  GOTO    SCAN
                   00227 ;
0080               00228 WRITX
0080 0553          00229  BSF     FLAGA,TR          ;SET TRIGGER FLAG
0081 0C40          00230  MOVLW   40H               ;IF XCOUNT > 64 THEN XCOUNT <-64
0082 0094          00231  SUBWF   XCOUNT,W
0083 0603          00232  BTFSC   STATUS,CA
0084 0A8D          00233  GOTO    WRITR
0085               00234 WRITS
0085 0776          00235  BTFSS   FLAGB,RI          ;LEFT / RIGHT ?
0086 0A90          00236  GOTO    WRITL
0087 0274          00237  COMF    XCOUNT, F         ;(RIGHT) NEG XCOUNT
0088 0294          00238  INCF    XCOUNT,W
0089               00239 WRITA
0089 002F          00240  MOVWF   DATA1
008A 0031          00241  MOVWF   DATA3
008B 0074          00242  CLRF    XCOUNT            ;RESET XCOUNT
008C 0A5D          00243  GOTO    SCANA
                   00244 ;
008D               00245 WRITR
008D 0C40          00246  MOVLW   40H               ;XCOUNT <- 64
008E 0034          00247  MOVWF   XCOUNT
008F 0A85          00248  GOTO    WRITS
                   00249 ;
0090               00250 WRITL
0090 0214          00251  MOVF    XCOUNT,W          ;(LEFT)
0091 0A89          00252  GOTO    WRITA
```


© 1997 Microchip Technology Inc.

```
                    00253 ;
0092                00254 WRITY
0092 0553           00255  BSF    FLAGA,TR          ;SET TRIGGER FLAG
0093 0C40           00256  MOVLW  40H               ;IF YCOUNT > 64 THEN YCOUNT <-64
0094 0095           00257  SUBWF  YCOUNT,W
0095 0603           00258  BTFSC  STATUS,CA
0096 0A9F           00259  GOTO   WRITV
0097                00260 WRITW
0097 0736           00261  BTFSS  FLAGB,UP           ;DOWN / UP ?
0098 0AA2           00262  GOTO   WRITD
0099 0275           00263  COMF   YCOUNT, F          ;(UP) NEG YCOUNT
009A 0295           00264  INCF   YCOUNT,W
009B                00265 WRITB
009B 0030           00266  MOVWF  DATA2
009C 0032           00267  MOVWF  DATA4
009D 0075           00268  CLRF   YCOUNT             ;RESET YCOUNT
009E 0A60           00269  GOTO   SCANB
                    00270 ;
009F                00271 WRITV
009F 0C40           00272  MOVLW  40H               ;YCOUNT <- 64
00A0 0035           00273  MOVWF  YCOUNT
00A1 0A97           00274  GOTO   WRITW
                    00275 ;
00A2                00276 WRITD
00A2 0215           00277  MOVF   YCOUNT,W          ;(DOWN)
00A3 0A9B           00278  GOTO   WRITB
                    00279 ;
                    00280 ;=========================================
                    00281 ;        RESET ENTRY
                    00282 ;=========================================
                    00283 ;
01FF                00284  ORG    777
01FF 0A45           00285  GOTO   INIT              ;JUMP TO PROGRAM STARTING
                    00286 ;
                    00287  END

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXX------------ ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   165
Program Memory Words Free:   347


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

A Comparison of 8-Bit Microcontrollers

Author: *Mark Palmer*
Microchip Technology Inc.

INTRODUCTION

The PIC16C5X/XX microcontrollers from Microchip Technology Inc., provide significant execution speed and code-compaction improvement over any other 8-bit microcontroller in its price range.

The superior performance of the PIC16C5X/XX microcontrollers can be attributed primarily to its RISC architecture. The PIC16C5X/XX devices employ a Harvard architecture (i.e., has separate program memory space and data memory space [8-bit wide data]). It also uses a two stage pipelining instruction fetch and execution. All instructions are executed in a single cycle (200 ns @ 20 MHz clock) except for program branches which take two cycles, and there are only 33 instructions to remember.

Separation of program and data space allows the instruction word to be optimized to any size (12-bit wide for PIC16C5X devices and 14-bit wide for PIC16CXX devices). This makes it possible, for example, to load an 8-bit immediate value in one cycle. First, because there is no conflict between instruction fetch and data fetch (as opposed to von Neumann architecture) and secondly because the instruction word is wide enough to hold the 8-bit data.

In the following sections we will compare the PIC16C5X/XX devices @ 20 MHz with:

- SGS-Thomson ST62 @ 8 MHz
- Motorola MC68HC05 @ 4.2 MHz
- Intel 8051 @ 20 MHz
- Zilog Z86CXX @ 12 MHz
- National COP800 @ 20 MHz

Several coding examples will be considered. While the comparisons are not entirely scientific, they will demonstrate to the reader the relative superior performance of the PIC16C5X/XX devices. The examples chosen are used frequently in microcontroller applications.

PACKING BINARY CODED DECIMAL (BCD)

This example will take two bytes in RAM or registers, each containing a BCD digit in the lower nibble and create a packed BCD data byte, which is stored back in the register or RAM location holding the low BCD digit.

PIC16C5X/XX

		Byte/Words	Cycles
SWAPF	REGHI,W	1	1
IORWF	REGLO	1	1
		2	2

0.4 µs

COP800

		Byte/Words	Cycles
X	A,[B+]	1	2
SWAP	A	1	1
OR	A,[B]	1	1
X	A,[B]	1	1
		4	5

B is pointing to the higher BCD digit initially. After auto-increment, it points to the lower BCD digit.

5 µs

ST62

		Byte/Words	Cycles
LD	A,REGHI	2	4
RLC	A	1	4
RLC	A	1	4
RLC	A	1	4
RLC	A	1	4
ADD	A,REGLO	2	4
LD	REGLO,A	2	4
		10	28

REGHI and REGLO are registers addressable by short direct addressing mode.

45.5 µs

MC68HC05

		Byte/Words	Cycles
LDA	REGHI	2	3
ROLA		1	3
ROLA		1	3
ROLA		1	3
ROLA		1	3
ADD	REGLO	2	3
STA	REGLO	2	4
		10	22

10.5 µs

Z86CXX

		Byte/Words	Cycles
SWAP	REGHI	2	8
OR	REGHI,REGLO	2	6
		4	14

REGHI and REGLO are addressable via the working register addressing mode.

5.33 µs

8051

		Byte/Words	Cycles
MOV	A,Rx	1	1
SWAP	A	1	1
ORL	A,Ry	1	1
MOV	Ry,A	1	1
		4	4

Register Rx contains higher BCD digit, Ry holds lower BCD digit.

2.4 µs

© 1997 Microchip Technology Inc.

LOOP CONTROL

This example is one of simple loop control where a register containing a loop count is decremented, tested for zero, and if not zero, then branched back to the beginning of the loop.

PIC16C5X/XX

		Byte/Words	Cycles
DECFSZ	COUNT	1	1/2
GOTO	BEG_LOOP	1	2/-
		2	3/2

0.6 µs/0.4 µs

COP800

		Byte/Words	Cycles
DRSZ	COUNT	1	3
JP	BEG_LOOP	1	3
		2	6

COUNT is Register (RAM F0h-FFh).

6 µs

ST62

		Byte/Words	Cycles
DEC	X	1	4
JRZ	BEG_LOOP	1	2
		2	6

9.75 µs

MC68HC05

		Byte/Words	Cycles
DECX		1	3
BEQ	BEG_LOOP	2	3
		3	6

2.86 µs

Z86CXX

		Byte/Words	Cycles
DJNZ	COUNT, BEG_LOOP	2	10/12

1.67 µs/2.0 µs

8051

		Byte/Words	Cycles
DJNZ	Rx, BEG_LOOP	2	2

1.2 µs

BIT TEST & BRANCH

This example tests a single bit in a register or a RAM location and makes a conditional branch. We assume that the MSb is tested and a branch is to be taken if the bit is set.

PIC16C5X/XX

		Byte/Words	Cycles
BTFSC	REG, 7	1	1/2
GOTO	NEWADD	1	2/-
		2	3/2

0.6 µs/0.4 µs

COP800

		Byte/Words	Cycles
IFBIT	7, [B]	1	1
JP	NEWADD	1	3
		2	4

B points to the memory location under test.

4 µs

ST62

		Byte/Words	Cycles
JRR	7, NEWADD	3	5

8.125 µs

MC68HC05

		Byte/Words	Cycles
BRCLR	7, NEWADD	3	5

2.38 µs

Z86CXX

		Byte/Words	Cycles
BTJRT	NEWADD, REG, 7	3	16/18

2.67 µs/3.0 µs

8051

		Byte/Words	Cycles
MOV	A, @Rx	1	1
JB	A.7, NEWADD	3	2
		4	3

Register Rx is assumed to be pointing to the memory location under test.

1.8 µs

© 1997 Microchip Technology Inc.

SHIFTING OUT 8-BIT DATA & CLOCK

We will now consider the task of serially shifting out an 8-bit data and clock. Data and clock outputs are generated under program control by toggling two output pins.

Data is transmitted on the rising edge of the clock. No attempt is made to make the clock output symmetrical in order to make the code efficient. Data out is guaranteed on the falling edge of the clock. These conditions are satisfactory for most applications.

PIC16C5X/XX

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
XMIT	MOVLW	08H	; Bit Count	1	1	1	
	MOVWF	BITCNT	;	1	1	1	
XM1	BCF	PORTB, 0	; 0 → Data Out Pin	1	1	1	
	BCF	PORTB, 1	; 0 → Clock Out Pin	1	1	1	
	RRF	XDATA	; Rotate Right thru Carry	1	1	1	
	BTFSC	STATUS, CARRY	; Test Carry Bit	1	2	1	
	BSF	PORTB, 0	; 1 → Data Out Pin	1	—	1	
	BSF	PORTB, 1	; 1 → Clock Out Pin	1	1	1	Loop
	DECFSZ	BITCNT	; Decrement Count ; Skip if Zero	1	1	1	
	GOTO	XM1	;	1	2	2	
	BCF	PORTC, 1	; 0 → Clock	1	1	1	
				11	74	74	

Transmit time is the same for 00h or FFh: 74 Tcyc = 14.8 µs. Note that there was no need to load the data into the Accumulator (W) since the PIC16C5X/XX can operate directly on file registers.

COP800

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
XMIT	LD	A, XDATA	; Load Data in Acc.	2	3	3	
	LD	BITCNT #08H	; Load Bit Count	2	3	3	
	LD	B, #D0H	; B Points to PORTL	2	3	3	
XM1	RBIT	0,[B]	; 0 → Clock	1	1	1	
	RBIT	1,[B]	; 0 → Data	1	1	1	
	RRCA		; Rotate A Right thru Carry	1	1	1	
	IFC		;	1	1	1	
	SBIT	1,[B]	; 1 → Data	1	—	1	Loop
	SBIT	0,[B]	; 0 → Clock	1	1	1	
	DRSZ	BITCNT	; Decrement Bit Count	1	3	3	
	JP	XM1	; and Go Back if ≠ 0	2	3	3	
	RBIT	0,[B]	;	1	3	3	
				16	100	108	

Accumulator (A) is first loaded with the data word. Transmit time is maximum for data = FFh; 105 Tcyc = 105 µs.

ST62

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
	LDI	A, #08	; Bit Count	2	4	4	
	LD	X, A	; Xmit Data	1	4	4	
	LD	A, W	; 0 → Clock	1	4	4	
XM1	RES	0, DRB	; 0 → Data	2	4	4	
	RES	1, DRB	;	2	4	4	
	SLA	A	;	2	4	4	
	JRNC	XM2	; 1 → Data	1	2	2	
XM2	SET	1, DRB	; 1 → CLK	2	—	4	Loop
	SET	0, DRB	;	2	4	4	
	DEC	X	;	1	4	4	
	JRNZ	XM1	;	1	2	2	
	RES	0, DRB	;	2	4	4	
			; 0 → Data	19	208	240	

Register W contains the Data Word.
Transmit time for FFh = 240 cycles = 390 µs.

SHIFTING OUT 8-BIT DATA & CLOCK (Cont.'d)

MC68HC05

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
XMIT	LDA	XDATA	; Load Xmit Data	2	3	3	
	LDX	#$08	; Load Bit Count	2	2	2	
XM1	BCLR	0, PORTB	; 0 → Clock	2	5	5	
	BCLR	1, PORTB	; 0 → Data	2	5	5	
	ROLA		;	1	3	3	
	BCC	XM2	;	2	3	3	
	BSET	1, PORTB	; 1 → Data	2	—	5	Loop
	BSET	0, PORTB	; 1 → Clock	2	5	5	
XM2	DECX		;	1	3	3	
	BNE	XM1	;	2	3	3	
	BCLR	0, PORTB	; 0 → Data	2	5	5	
				20	226	266	

Transmit time is maximum for transmitting FFh = 266 cycles = 126.7 µs.

Z86CXX

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
XMIT	LD	COUNT, #8	; Load Bit Count	3	10	10	
	AND	P2, #%FC	; 0 → Data, Clock	3	6	6	
XM1	RRC	XDATA	;	2	6	6	
	JR	NC, XM2	;	2	12	10	
	OR	P2, #01	; 1 → Data	3	—	10	Loop
XM2	OR	P2, #02	; 1 → Clock	3	10	10	
	DJNZ	COUNT, XM1	;	2	12	12	
	AND	P2, #%FC	; 0 → Clock, Data	3	10	10	
				21	348	412	

Transmit time is maximum for transmitting FFh = 412 cycles = 68.67 µs.

8051

				Byte /Words	Cycles Xmit 00h	Cycles Xmit FFh	
MXIT	MOV	A, @R0	; R0 Points to Data Word	1	1	1	
	MOV	R1, #08H	; Load Bit Count	2	1	1	
XM1	ANL	PORT1, #0FCH	; 0 → Data, Clock	3	2	2	
	RRC	A	; Rotate Right A thru Carry	1	1	1	
	JNC	XM2	;	2	2	2	
	SETB	PORT1, 0	; 1 → Data	2	—	1	Loop
XM2	SETB	PORT1, 1	; 1 → Clock	2	1	1	
	DJNZ	R1, XM1	; Decrement Count	2	2	2	
				15	66	74	

Transmit time is maximum for transmitting FFh = 74 cycles = 44.4 µs.

© 1997 Microchip Technology Inc.

SOFTWARE TIMER

Microcontrollers quite often need to implement time delays. Debouncing key input, pulse width modulation, and phase angle control are just a few examples. Implementing a 10 ms time delay loop subroutine will be considered in this section.

PIC16C5X/XX

				Byte/Words	Cycles
DELAY	MOVLW	41H	; 10 ms Delay Loop	1	1
	MOVWF	COUNT2	;	1	1
	CLRF	COUNT1	;	1	1
LOOP	INCFSZ	COUNT1	; This inner Loop will be	1	2/1
	GOTO	LOOP	; Executed 256 Times	1	2
	DECFSZ	COUNT2	;	1	2/1
	GOTO	LOOP	;	1	2
	RET		;	1	2
				8	

Execution time for the routine = 5 + (255 x 3 + 5) x 65 = 20025 Tcyc = 10.011 ms. The PIC16C5X/XX can implement delay times very precisely (when necessary) because of its fine instruction cycle resolution.

COP800

				Byte/Words	Cycles
DELAY	LD	COUNT1, #0BH	; 10 ms Delay Loop	2	3
	LD	B, #0EH	;	1	1
LOOP	DRSZ	B	;	1	1
	JP	LOOP	;	1	1
	DRSZ	COUNT1	;	1	1
	JP	LOOP	;	1	1
	RET		;	1	5
				8	

Execution time for the routine = (6N2 + 6) N1 + 9 cycles. Here N1 = 0Bh and N2 = 0Eh, which gives us: 999 Tcyc = 9.99 ms.

ST62

				Byte/Words	Cycles
	LDI	A, #FF		2	4
	LD	X, A	; LOOP1 Count	1	4
	LDI	A, #04		2	4
	LD	Y, A	; LOOP2 Count	1	4
LOOP	DEC	X	; 0 CLK	1	4
	JRNZ	LOOP	;	1	2
	DEC	Y	; 0 CLK	1	4
	JRNZ	LOOP		1	2
				10	

Execution time for the subroutine = (6N1 + 6) N2 + 16 cycles, where N1 = FFh, N2 = 04h gives us 10.01 ms.

© 1997 Microchip Technology Inc.

SOFTWARE TIMER (Cont.'d)

MC68HC05

				Byte/Words	Cycles
DELAY	LDX	$2D	; 10 ms Delay Loop	2	2
	LDX	$5C	;	2	2
LOOP	DECA		;	1	3
	BNE		;	2	2
	DECX	LOOP	;	1	3
	BNE		;	2	2
	RTS	LOOP	;	1	6
				11	

Execution time for the subroutine = (5 x N1 + 5)N2 + 10, with N1 = 2Dh, N2 = 5Ch, time delay = 10.081 ms.

Z86CXX

				Byte/Words	Cycles
DELAY	LD	COUNT1, #%61	; 10 ms Delay Loop	2	6
	LD	COUNT2, #%33	;	2	6
LOOP	DJNZ	COUNT1, LOOP	;	2	10/12
	DJNZ	COUNT2, LOOP	;	2	10/12
	RET		;	1	14
				9	

Total execution time = (12N1 +10)N2, with N1 = 61h, N2 = 33h, time delay = 59976 cycles = 9.979 ms.

8051

				Byte/Words	Cycles
DELAY	MOV	COUNT1, #21H	; 10 ms Delay Loop	2	1
LOOP1	MOV	COUNT2, #FBH	;	2	1
LOOP2	DJNZ	COUNT2, LOOP2	;	3	2
	DJNZ	COUNT1, LOOP1	;	3	2
	RET		;	1	2
				11	

Execution time for the subroutine = (2N1 + 3)N2 + 3 cycles. Where N1 = FBh, N2 = 21h, which gives us: 16668 cycles = 10.0008 ms.

© 1997 Microchip Technology Inc.

SUMMARY

Table 1 summarizes code sizes for different microcontrollers. The overall relative code size number is an average of the individual relative code sizes. Given that the program word size of the PIC16C5X/XX is 12- or 14-bit (compared to an 8-bit program memory of all the other microcontrollers), a compaction of 1.5 is expected. Clearly, the PIC16C5X/XX meets this compaction (except for the COP800) and exceeds the compaction ratio in most comparisons.

Table 2 summarizes relative execution speed. The overall speed is an average of relative speed numbers. For example, the COP800 will, on average, exhibit 27% of the code execution speed of a PIC16C5X/XX devices. In other words, a PIC16C5X/XX will be (1/0.27) 3.7 times faster than a COP800 on average.

TABLE 1: COMPARISON OF CODE EFFICIENCY*

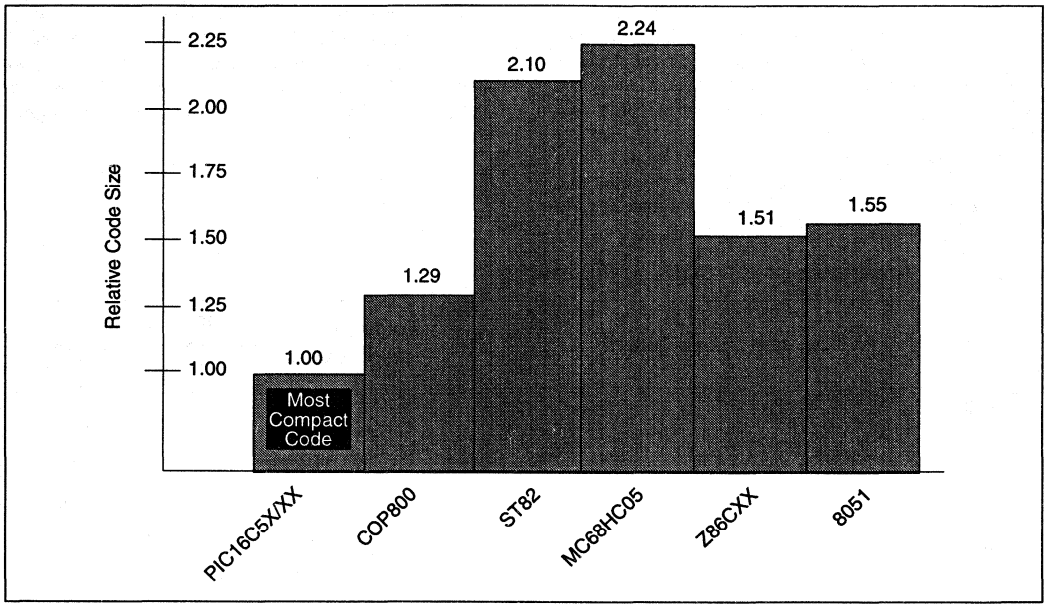
Device	Packing BCD	Loop Control	Bit Test & Branch	8-Bit Sync Transmission	10 ms Software Timer	Overall
COP800	4 2.00	2 1.00	2 1.00	16 1.46	8 1.00	1.29
ST62	10 5.00	2 1.00	3 1.50	19 1.73	10 1.25	2.10
MC68HC05	10 5.00	3 1.50	3 1.50	20 1.82	11 1.38	2.24
Z86CXX	4 2.00	2 1.00	3 1.50	21 1.91	9 1.125	1.51
8051	4 2.00	2 1.00	4 2.00	15 1.36	11 1.375	1.547
PIC16C5X/XX	2	2	2	11	8	1.00

* In each box, the top number is the number of program memory locations required to code the application. The bottom number is relative code size compared to the PIC16C5X/XX:

$$\frac{\text{\# program memory locations for other microcontroller}}{\text{\# program memory locations for the PIC16C5X/XX}}$$

© 1997 Microchip Technology Inc.

FIGURE 1: CODE SIZE COMPARISON



© 1997 Microchip Technology Inc.

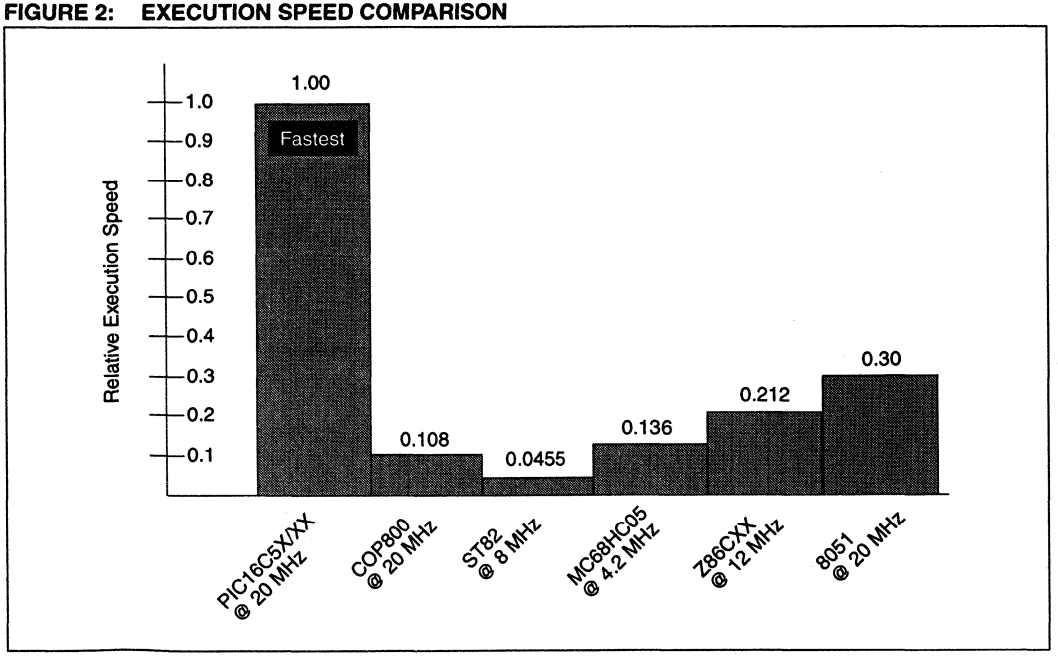
TABLE 2: COMPARISON OF EXECUTION SPEED

Device	Packing BCD	Loop Control	Bit Test & Branch	8-Bit Sync Transmission	10 ms Software Timer	Overall
COP800 @ 20 MHz	5 µs 0.08	6 µs 0.0832	4 µs 0.1252	105 µs 0.1408	—	0.108
ST62 @ 8 MHz	45.5 µs 0.0088	9.75 µs 0.0615	8.125 µs 0.0738	390 µs 0.0329	—	0.0455
MC68HC05 @ 4.2 MHz	10.05 µs 0.038	2.86 µs 0.1748	2.38 µs 0.21	126.7 µs 0.1168	—	0.136
Z86CXX @ 12 MHz	2.33 µs 0.172	1.835 µs 0.272	2.835µs 0.176	68.67 µs 0.224	—	0.212
8051 @ 20 MHz	2.4 µs 0.1666	1.2 µs 0.4166	1.8 µs 0.277	44.4 µs 0.33	—	0.30
PIC16C5X/XX @ 20 MHz	0.4 µs	0.6/0.4 µs	0.6/0.4 µs	14.8 µs	—	1.00

* In each box, the top number is the time required to execute the example code, while the bottom number is a measure of relative performance compared to the PIC16C5X/XX.

$$\frac{\text{time required to execute code by the PIC16C5X/XX}}{\text{time required to execute code by other microcontroller}}$$

FIGURE 2: EXECUTION SPEED COMPARISON



NOTES:

© 1997 Microchip Technology Inc.

Interfacing to AC Power Lines

Author: *Doug Cox*
Microchip Technology Inc.

INTRODUCTION

This application note describes a simple method for measuring parameters from the AC power line. Parameters such as zero crossing, frequency, and relative phase can be measured. This method is useful for measurements on 50, 60, and 400 Hz power systems with voltages up to several hundred volts. The method requires only one external component, a resistor, and is more reliable than previously published methods using capacitors or bulky, expensive transformers.

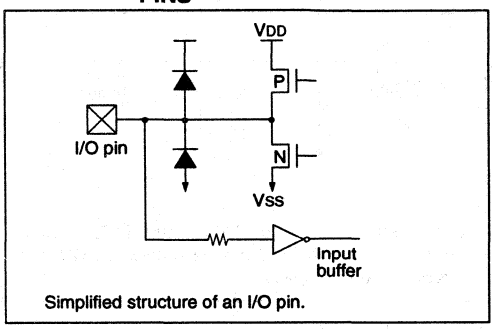
APPLICATIONS

This measurement method can be used in any application where power line parameters are used for system measurements or control. Typical applications are; switch timing (what part of the power cycle should the system be activated), power factor correction, power measurement, and power line monitor. An additional application is to generate timing or clock functions using the relatively stable power line frequency. This method is also useful for calibrating oscillator frequency for accurate timing measurements when an inaccurate reference such as an RC oscillator is used to clock the PIC16C5X.

THEORY OF OPERATION

This application takes advantage of the input static protection circuitry that exists on all I/O pins of a CMOS PIC16C5X. These protection circuits are designed to short the inputs to the power supplies when a large overvoltage is applied, thus protecting the chip from static electricity spikes. On PIC16C5X microcontrollers, this protection circuit is two large P-N diodes on each input (Figure 1). These diodes will short any voltage higher than VDD to the VDD supply and any voltage less than VSS to the VSS supply. They can take several milliamps of current without any damage to the chip. High voltages can be applied directly to the chip inputs as long as they are current limited.

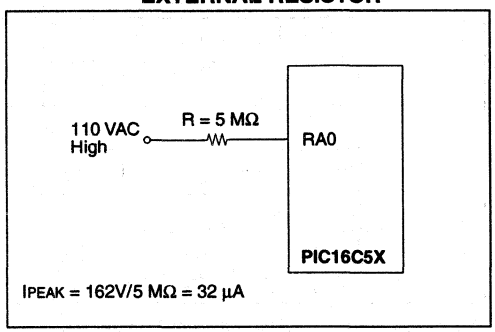
FIGURE 1: PIC16C5X SERIES INPUT PROTECTION CIRCUIT ON I/O PINS



Simplified structure of an I/O pin.

The least expensive method to limit current is to use a high value resistor. A block diagram is shown in Figure 2. The power line voltage is current limited by the resistor and then clamped by the input protection diodes internal to the PIC16C5X. A typical input waveform is shown in Figure 5. A 115 VAC, 60 cycle sine wave will traverse from 0 to 2V in 32 µs, therefore a typical threshold of 2V on a PIC16C5X I/O port will permit zero crossing detection accuracy of about 30 µs. If the typical capacitance on an I/O pin is 5 pF, then R should be (t = RC) 6 MΩ or less for best zero crossing accuracy. A 5 MΩ resistor with 115 VAC applied to it will limit current to 32 µA, a value which is well within the safety margin of the PIC16C5X.

FIGURE 2: LIMITING CURRENT USING AN EXTERNAL RESISTOR

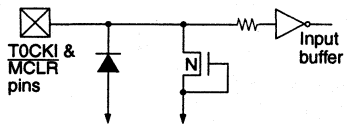


IPEAK = 162V/5 MΩ = 32 µA

© 1997 Microchip Technology Inc.

The user needs to be aware that the circuit required to connect the T0CKI input to an AC power line is slightly different than the other I/O pins. Each of the I/O pins has two diodes for input protection whereas the T0CKI pin has only one protection diode connected to Vss (Figure 3). Therefore, it is necessary to connect a diode externally between the T0CKI pin and VDD in order to clamp the voltage on the T0CKI pin to VDD + 0.6V (approximately). See Figure 4. It is also recommended that resistor R be at least 2 MΩ.

FIGURE 3: INPUT STRUCTURE OF $\overline{MCLR}$ AND T0CKI PINS



Simplified structure of T0CKI and $\overline{MCLR}$ pins. Grounded gate NMOS device provides ESD and overvoltage protection.

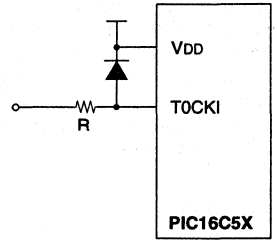
RELIABILITY

Reliability of production devices that are directly connected to AC power is always a concern. Two failure modes are possible. First, the series resistor of Figure 1 might fail short, destroying the microcontroller.

A short is the most unlikely failure mode for a resistor, and resistors are more reliable than transformers or capacitors, which are the alternate components for measuring line parameters. This reliability can be enhanced even further by using two resistors in series. Both would have to fail short to cause catastrophic failure, a very unlikely event.

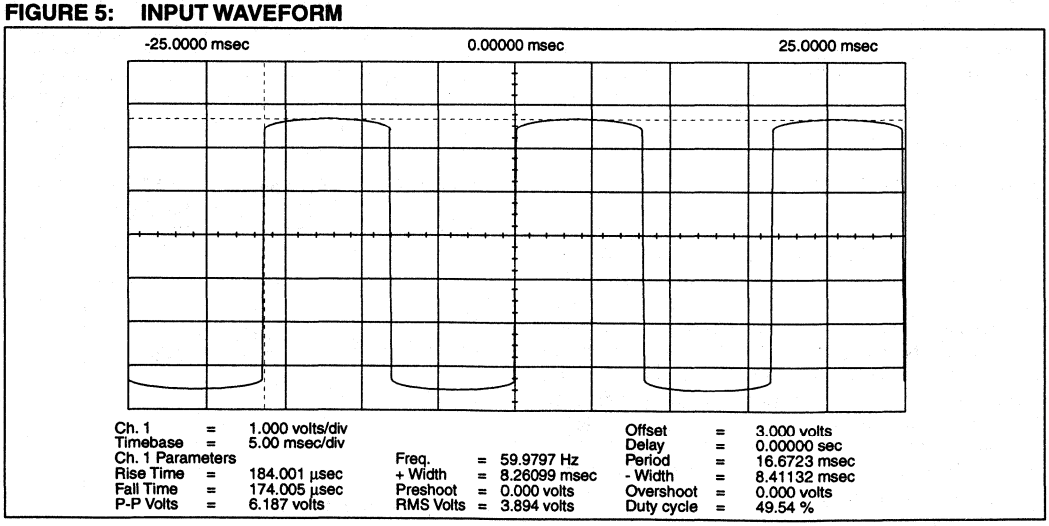
The second possible failure mode is that excessive injection of current into the PIC16C5X might cause the protection diode to open. This would allow the input to go to power line peak voltage (162V) and short the input transistor gate oxide, causing device failure. The maximum continuous injection current into an I/O pin is specified at ± 500 µA.

FIGURE 4: CONNECTING AN AC POWER LINE TO THE T0CKI PIN



Connecting to TMR0 input. R ≥ 2 MΩ

FIGURE 5: INPUT WAVEFORM



Waveform at part pin (RA0) •R = 100k; Line: 60 Hz, 110V

© 1997 Microchip Technology Inc.

Power-up Considerations

Author: Sumit Mitra
Microchip Technology Inc.

INTRODUCTION

When powering up any microcontroller it is necessary for the power supply voltage to traverse voltage ranges, where the device is not guaranteed to operate, before the power supply voltage reaches its final state. Since some circuits on the device (logic) will start operating at voltage levels lower than other circuits on the chip (memory), the device may power-up in an unknown state. To guarantee that the device starts up in a known state, it must contain a power-up reset circuit. PIC16C5X microcontrollers are equipped with on-chip Power-on Reset circuitry, which eliminates the need for external reset logic. This circuit will function in most power-up situations where VDD rise time is fast enough (50 ms or less). This application note describes the typical power-up sequence for PIC16C5X microcontrollers. Methods of assuring reset on power-up and after a brown-out are discussed and simple, low-cost external solutions are discussed for power-up situations where the PIC16C5X's internal circuitry cannot provide the reset.

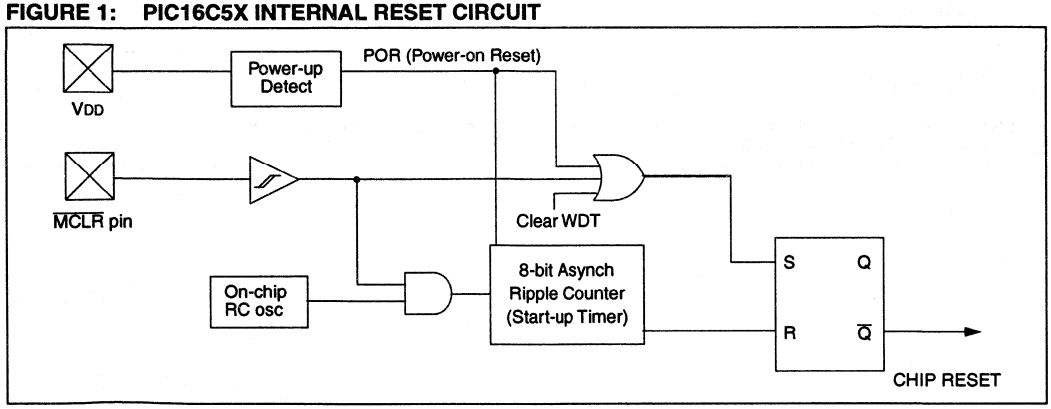
POWER-UP SEQUENCE

The PIC16C5X incorporates complex Power-on Reset (POR) circuitry on-chip which provides a solid, reliable internal chip reset for most power-up situations. To use this feature, the user merely needs to tie the $\overline{MCLR}$ pin to the VDD pin. A simplified block diagram of the on-chip reset circuitry is shown in Figure 1. On power-up, the reset latch and the start-up timer are reset to appropriate states by the Power-on Reset (POR). The start-up timer will begin counting once it detects $\overline{MCLR}$ to be high (i.e., external chip reset goes inactive). After the time-out period, which is typically 18 ms long, the timer will reset the reset latch and thus end the on-chip reset signal.

Figure 3 and Figure 4 are two power-up situations with relative fast rise time on VDD. In Figure 2, VDD is stable when $\overline{MCLR}$ is brought high (i.e., reset pulse is being provided by external source). The chip actually comes out of reset about TOST ms after that, where TOST = Time of the Oscillator Start-up Timer. (The timer is called Oscillator Start-up Timer because the time-out was incorporated primarily to allow the crystal oscillator to stabilize on power-up). In Figure 3, the $\overline{MCLR}$ and VDD are tied together and clearly the on-chip reset mechanism is being used. VDD is stable before the start-up timer expires and there is no problem with proper reset.

In Figure 4, where the VDD rise time is much greater than TOST (typically 18 ms) this is clearly a potentially problematic situation. The POR pulse comes when VDD is about 1.5V. Most CMOS logic, including the start-up timer, starts functioning between 1.5V to 2.0V. When the start-up timer times out, the chip reset is ended and the chip attempts to execute. If by this time VDD has reached the VDD min. value, then all circuits are guaranteed to function correctly and power-up reset is successful. If, however, the VDD slope was too slow and had not reached VDD min., then the chip is not guaranteed to function properly.

FIGURE 1: PIC16C5X INTERNAL RESET CIRCUIT



© 1997 Microchip Technology Inc.

FIGURE 2: EXTERNAL RESET PULSE

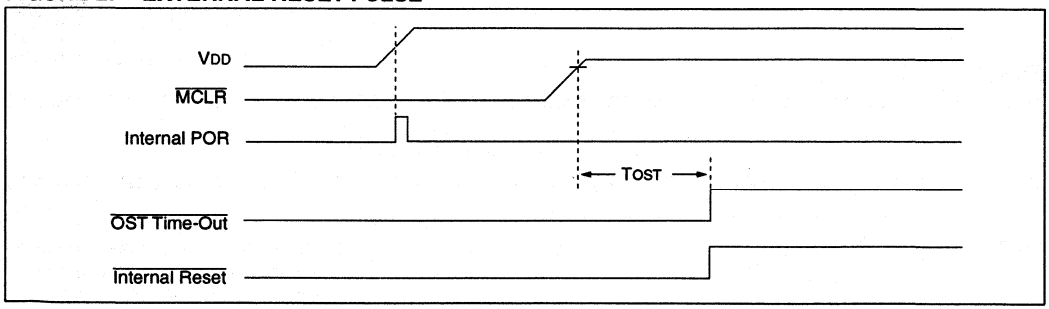


FIGURE 3: INTERNAL RESET (VDD AND MCLR TIED TOGETHER)

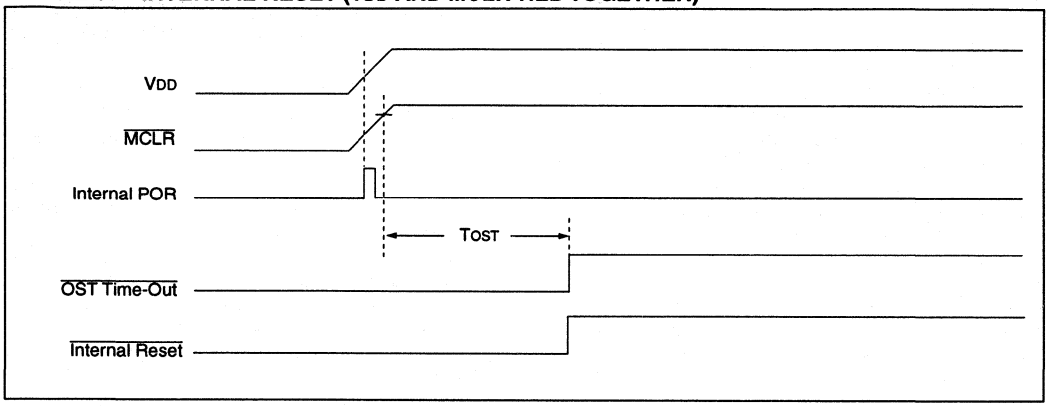
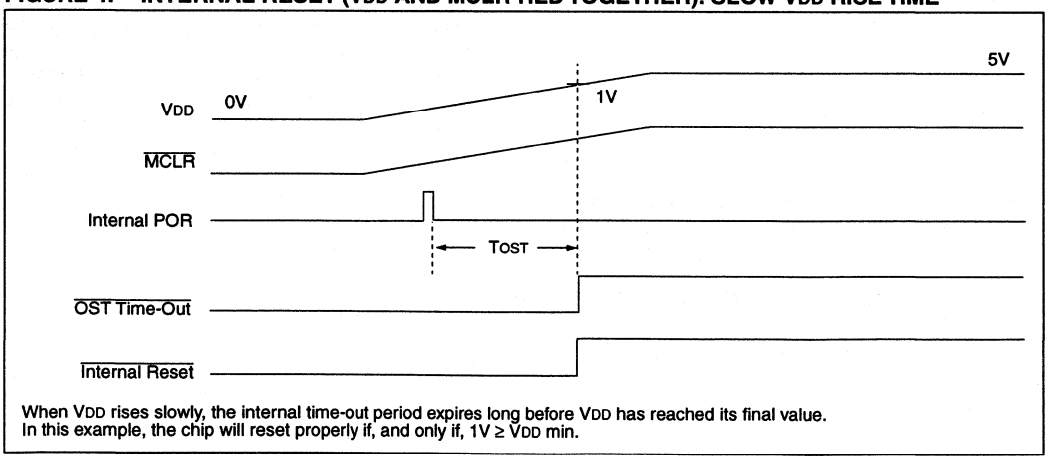


FIGURE 4: INTERNAL RESET (VDD AND MCLR TIED TOGETHER): SLOW VDD RISE TIME



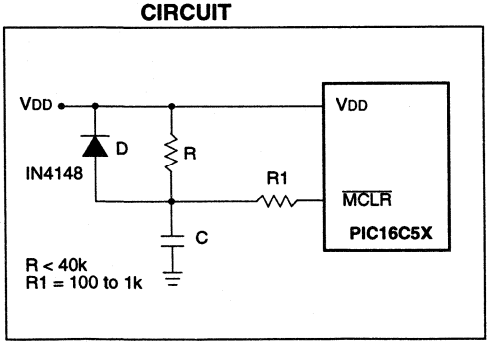
When VDD rises slowly, the internal time-out period expires long before VDD has reached its final value.
In this example, the chip will reset properly if, and only if, 1V ≥ VDD min.

© 1997 Microchip Technology Inc.

EXTERNAL POWER-ON RESET CIRCUIT

To use power supplies with slow rise times it is necessary to use an external power-on reset circuit such as the one shown in Figure 5. This circuit uses an external RC to generate the reset pulse. The time constant of the RC should be long enough to guarantee that the reset pulse is still present until VDD has reached VDD min. R should be 40k or less to guarantee that $\overline{MCLR}$ will pull to within 0.2 volts of VDD (since the leakage spec on $\overline{MCLR}$ is ±5 µA, a resistor larger than 40k may cause input high voltage on this pin to be less than VDD – 0.2V, the required spec). The diode *(D)* is used to rapidly discharge the capacitor on power-down. This is very important as a power-up reset pulse is needed after a short power-down (less than the time constant of RC) or after a power spike. The resistor *(R1)* protects against a high current flowing into the *$\overline{MCLR}$* pin from fully charged capacitor *(C)*, in the event the $\overline{MCLR}$ pin breakdown is induced via ESD (electrostatic discharge) or EOS (electrical overstress). The circuit, however, does not protect against brown-out situations where power does not drop to zero, but merely dips below VDD min. In such a situation, voltage at the $\overline{MCLR}$ pin will not go low enough (i.e., below VIL) to guarantee a reset pulse. The following section presents an example circuit to protect against such brown-outs.

FIGURE 5: EXTERNAL POWER-ON RESET CIRCUIT



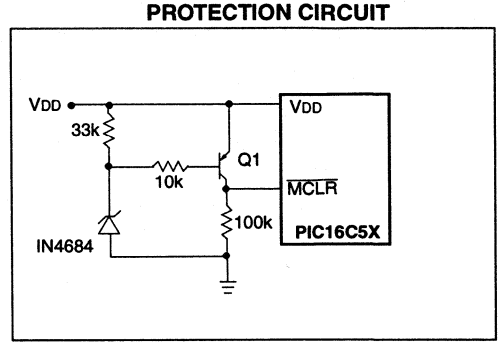
BROWN-OUT PROTECTION

In many applications it is necessary to guarantee a reset pulse whenever VDD is less than VDD min. This can be accomplished using a brown-out protection circuit such as the one shown in Figure 6. This is a simple circuit that causes a reset pulse whenever VDD drops below the zener diode voltage plus the VBE

$$V_{BE} = V_{DD} \bullet R1 / (R1 + R2)$$

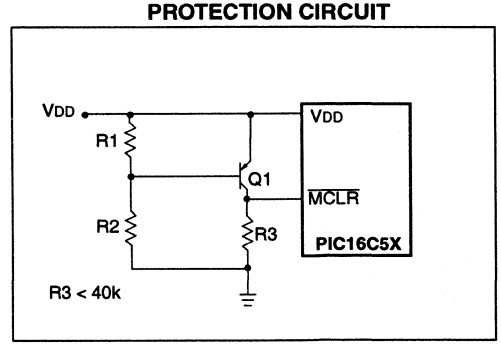
of *Q1*. A 3.3V zener will produce a reset pulse whenever VDD drops below about 4V. This circuit has a typical accuracy of about ±100 mV.

FIGURE 6: EXTERNAL BROWN-OUT PROTECTION CIRCUIT



A less expensive, albeit less precise, brownout circuit is shown in Figure 7. Transistor *Q1* turns off when VBE falls below 0.7V allowing *R3* to pull down the $\overline{MCLR}$ input.

FIGURE 7: EXTERNAL BROWN-OUT PROTECTION CIRCUIT



NOTES:

© 1997 Microchip Technology Inc.

Software Stack Management

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

The PIC16C5X has a stack which is only 2 deep, and, as a result, only two nested calls can be made (i.e., only one call within a call routine). If more than two levels of subroutine nesting is required, this application note can be used to implement a stack manager to handle the flow of the calls.

Note: Since the amount of RAM on the PIC16C5X is limited, it would be prudent to determine the maximum number of nested calls which have to be made in a program and define the stack length appropriately.

IMPLEMENTATION

This application note implements a 5-deep stack, so 5 nested calls can be made without overflowing the stack. `NCALL` is defined as a MACRO which will be used instead of the mnemonic `CALL`, when a subroutine call is made. The `NCALL` routine, "pushes" the return PC value on the "stack" and then executes the called subroutine. At the end of the subroutine, instead of using the `RETLW k` instruction, a `GOTO RETURN` is executed, where `RETURN` is a routine which "pops" the return PC value from the "stack" and resumes the normal flow of the program.

Note: Since Software Stack Management utilizes the FSR register, and indirect addressing, the user should restore the "original" values to the FSR register if it is used elsewhere in the program.

The routines, as described in this application note, will work only if the called routine is within the first 256 words for each program. If the user desires to branch over to the other low 256-byte program pages, as in the PIC16C57, then the status byte should be saved along with the PC.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: SOFTWARE STACK MANAGEMENT SAMPLE PROGRAM

```
MPASM 01.40 Released               SM.ASM   1-16-1997  13:03:22         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************
                     00004 ;       sm.asm:
                     00005 ;       Routine, demonstrating how to implement a stack
                     00006 ;       manager capable of handling more than 2
                     00007 ;       subsequent subroutine calls.
                     00008 ;       Note: Since this is a demo, NOP has been used
                     00009 ;       where normally the body of the subroutine would
                     00010 ;       reside.
                     00011 ;
                     00012 ;       Program:          SM.ASM
                     00013 ;       Revision Date:
                     00014 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00015 ;
                     00016 ;*******************************************************************
                     00017 ;
                     00018 ;
  00000002           00019 PC      EQU     2
  00000004           00020 FSR     EQU     4
                     00021 ;
  00000001           00022 F       EQU     1
                     00023 ;
0008                 00024         ORG     8
0008                 00025 STACK   RES     5       ;define stack size = 5
                     00026 ;
01FF                 00027         ORG     01FF
01FF 0A07            00028         GOTO    START
                     00029 ;
0000                 00030         ORG     0
                     00031 ;
0000 0C08            00032 INIT    MOVLW   STACK   ;load "stack" as indirect pointer
0001 0024            00033         MOVWF   FSR     ;       /
0002 0A07            00034         GOTO    START   ;      /
                     00035 ;
                     00036 ;*****************************************************
                     00037 ;define NCALL as a MACRO used instead of the
                     00038 ;mnemonic CALL.
                     00039 ;
                     00040 NCALL   MACRO   LABEL
                     00041         MOVF    PC,W    ;save PC on "stack"
                     00042         MOVWF   0       ;       /
                     00043         INCF    FSR, F  ;Inc. "stack"  pointer.
                     00044         GOTO    LABEL   ;jump to routine
                     00045         ENDM
                     00046 ;
                     00047 ;return from subroutine NCALL
                     00048 ;
0003 00E4            00049 RTN     DECF    FSR, F  ;point to last "stack" location
0004 0C03            00050         MOVLW   3       ;add 3 and output value from FSR
0005 01C0            00051         ADDWF   0,W     ;       /
0006 0022            00052         MOVWF   PC      ;load in PC as next executable
                     00053                         ;instruction
                     00054 ;
```


© 1997 Microchip Technology Inc.

```
                    00055 ;*******************************************************
                    00056 ;
                    00057     PAGE
                    00058 ;
0007 0000           00059 START   NOP
                    00060         NCALL   TOM
0008 0202               M         MOVF    PC,W    ;save PC on "stack"
0009 0020               M         MOVWF   0       ;       /
000A 02A4               M         INCF    FSR, F  ;Inc. "stack"  pointer.
000B 0A0F               M         GOTO    TOM     ;jump to routine
000C 0000           00061         NOP             ;body of main routine
000D 0000           00062         NOP             ;       /
000E 0003           00063         SLEEP
                    00064 ;
000F 0000           00065 TOM     NOP
                    00066         NCALL   DICK
0010 0202               M         MOVF    PC,W    ;save PC on "stack"
0011 0020               M         MOVWF   0       ;       /
0012 02A4               M         INCF    FSR, F  ;Inc. "stack"  pointer.
0013 0A16               M         GOTO    DICK    ;jump to routine
0014 0000           00067         NOP             ;body of routine TOM
0015 0A03           00068         GOTO    RTN
                    00069 ;
0016 0000           00070 DICK    NOP
                    00071         NCALL   HARRY
0017 0202               M         MOVF    PC,W    ;save PC on "stack"
0018 0020               M         MOVWF   0       ;       /
0019 02A4               M         INCF    FSR, F  ;Inc. "stack"  pointer.
001A 0A1D               M         GOTO    HARRY   ;jump to routine
001B 0000           00072         NOP             ;body of routine DICK
001C 0A03           00073         GOTO    RTN
                    00074 ;
001D 0000           00075 HARRY   NOP             ;body of routine HARRY
001E 0000           00076         NOP             ;       /
001F 0A03           00077         GOTO    RTN
                    00078 ;
                    00079 ;
                    00080         END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:    33
Program Memory Words Free:   479


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Implementing Wake-up on Key Stroke

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

In certain applications, the PIC16CXXX is exercised only when a key is pressed, (e.g., remote keyless entry). In such applications, the battery life can be extended by putting the PIC16CXXX to sleep during inactive states and when a key is pressed, the PIC16CXXX wakes up, performs the task, and then goes back to sleep.

IMPLEMENTATION

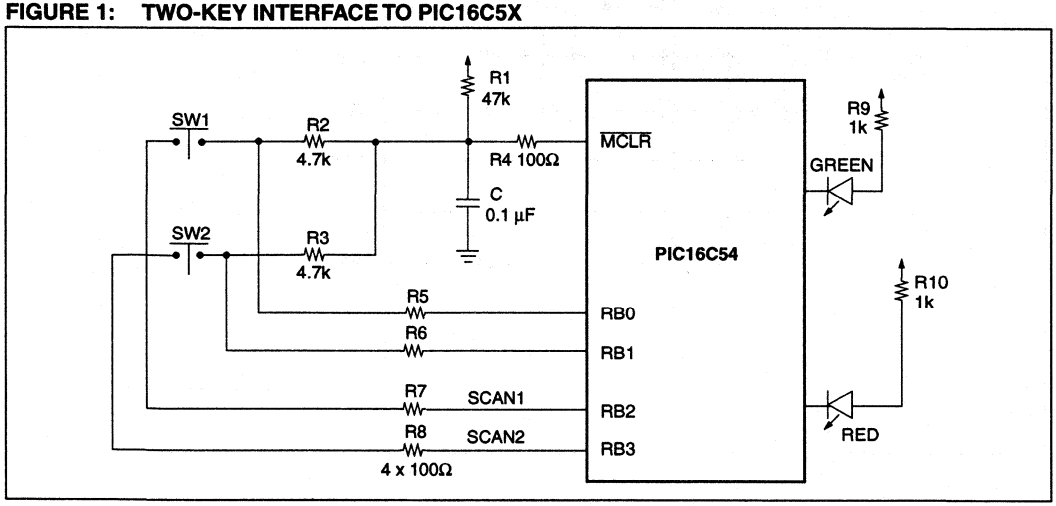
The circuit, shown Figure 1, depicts an application with two keys. The PIC16C54 is normally in SLEEP mode consuming very little operating current. If either of the two keys is pressed, the PIC16C5X "wakes up", scans the keys and turns on one or both of the LED's. When SW1 is pressed, the green LED is lit and when SW2 is pressed the red LED is lit. The LED's are used purely for demonstration purposes. In real life applications, a transmission would be completed before putting the PIC16C5X back to sleep. This example can be extended to handle more than two keys.

In sleep mode, the scan outputs (SCAN1 and SCAN2) are both set to a low logic level. In this state, the capacitor C is fully charged and a high logic level is present at the $\overline{MCLR}$ pin. When a key is pressed, C discharges through either R2 or R3 (depending on which switch is being pressed). The voltage across C falls rapidly (approx. 1 ms), causing a low level at the $\overline{MCLR}$ pin, which in turn causes the device to wake-up and enter its reset state. In reset, the SCAN1 and SCAN2 outputs default to a hi-impedance mode, which blocks the discharge path for capacitor C which it charges to a high level through resistor R1. Note that the RC values have been chosen such that the discharge and charge cycles times are less than the reset time for the device (approx. 18 ms), and certainly far less than the minimum duration of a key-press (approx. 50-100 ms).

After the reset cycle is completed, the code execution momentarily takes the SCAN1 and SCAN2 outputs low in order to sample the key stroke(s). This does not cause the capacitor to discharge since the duration of the low is of the order of 10 µs.

Once the keystroke function has been executed, the program loops until the key has been released, sets the SCAN1 and SCAN2 outputs low and "goes back to sleep". Resistors R4-R8 are not required for functionality, but are recommended to provide protection from electrostatic discharge (ESD). Switches SW1 and SW2, when pressed, may pass ESD to the PIC16C54.

FIGURE 1: TWO-KEY INTERFACE TO PIC16C5X



© 1997 Microchip Technology Inc.

FIGURE 2: TWO KEY SCAN/WAKE-UP TIMING DIAGRAM

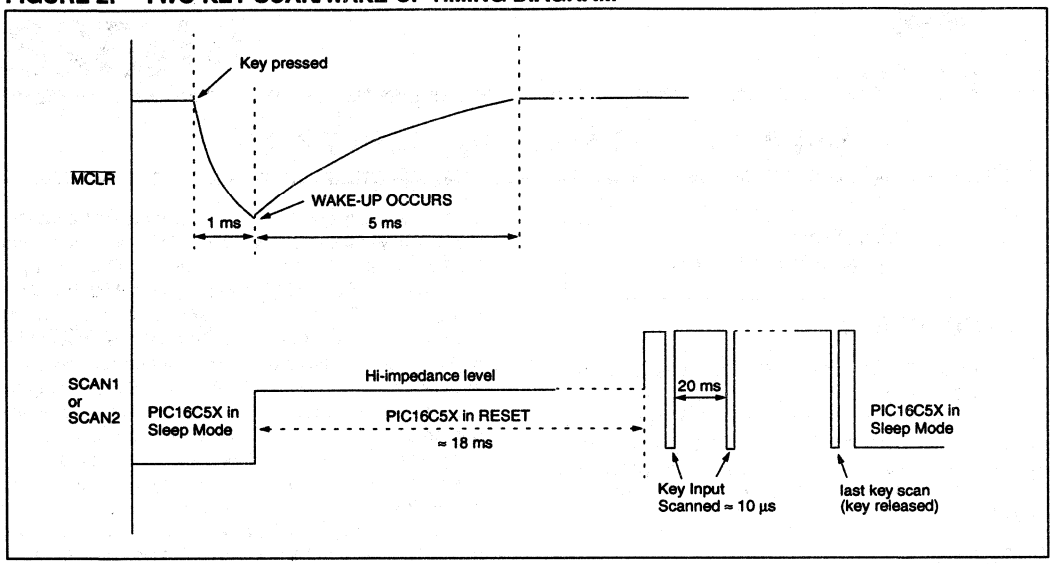
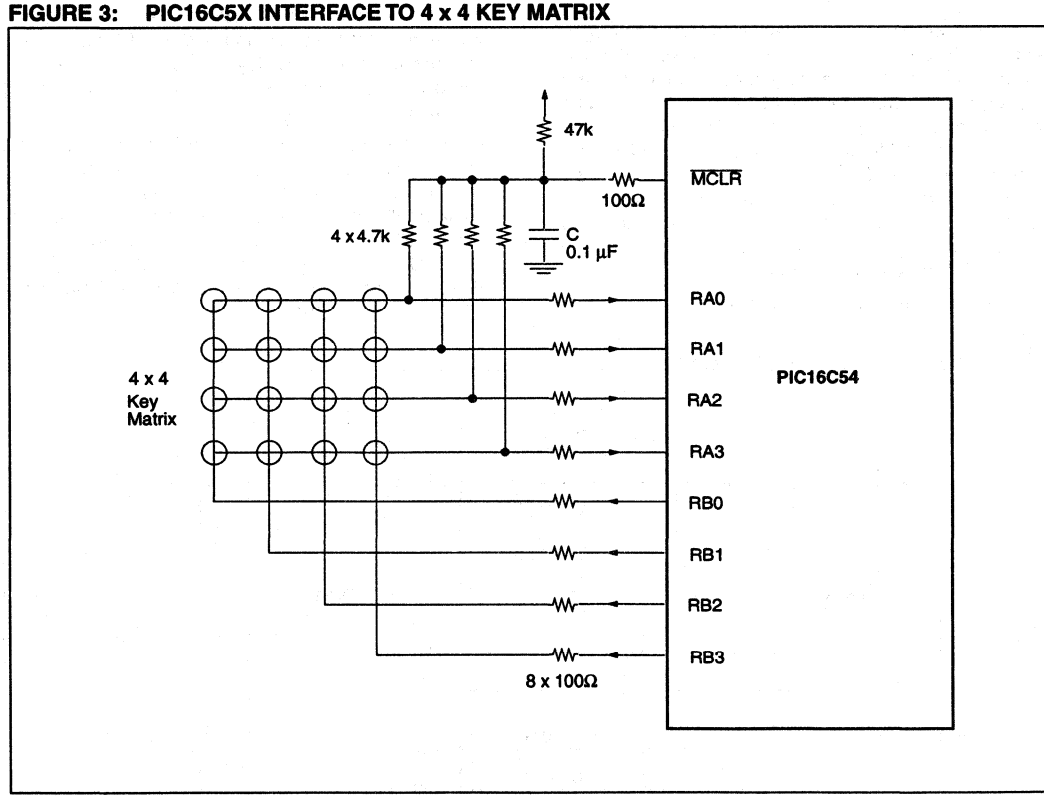


FIGURE 3: PIC16C5X INTERFACE TO 4 x 4 KEY MATRIX



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: KEY STROKE WAKE-UP SAMPLE PROGRAM

```
MPASM 01.40 Released               WU.ASM   1-16-1997  13:05:36          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001                    TITLE        "Key Stroke Wake Up"
                     00002     LIST  P = 16C54
                     00003 ;*****************************************************************
                     00004 ;       Program demonstrating key stroke wake up for
                     00005 ;       the PIC16CXXX. Program has been implemented for
                     00006 ;       two keys, but can be extended for more keys.
                     00007 ;       When SW1 is pressed a green LED lights up.
                     00008 ;       When SW2 is pressed a red LED lights up.
                     00009 ;
                     00010 ;       Program:          WU.ASM
                     00011 ;       Revision Date:
                     00012 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00013 ;
                     00014 ;*****************************************************************
                     00015 ;
                     00016 ;
                     00017 ; Define equates
                     00018 ;
  00000002           00019 PC        EQU    2
  00000006           00020 PORT_B    EQU    6
  00000002           00021 SCAN1     EQU    2
  00000003           00022 SCAN2     EQU    3
  00000000           00023 SW1       EQU    0
  00000001           00024 SW2       EQU    1
  00000004           00025 GRN_LED   EQU    4
  00000005           00026 RED_LED   EQU    5
  00000014           00027 MSEC_20   EQU    D'20'
  00000008           00028 DB1       EQU    8
  00000008           00029 GP        EQU    8
  00000009           00030 DB2       EQU    9
                     00031 ;
  00000001           00032 F         EQU    1
                     00033 ;
                     00034 ;PORT_B ASSIGNMENTS:
                     00035 ;       0 --> SW1        INPUT
                     00036 ;       1 --> SW2        INPUT
                     00037 ;       2 --> SCAN1      OUTPUT
                     00038 ;       3 --> SCAN2      OUTPUT
                     00039 ;       4 --> GRN_LED    OUTPUT
                     00040 ;       5 --> RED_LED    OUTPUT
                     00041 ;     6&7 --> ASSIGNED AS DUMMY OUTPUTS
                     00042     PAGE
                     00043 ;
                     00044 ;
0000                 00045         ORG      0
                     00046 ;
0000                 00047 START
0000 0910            00048         CALL     INIT_PORT_B      ;INITIALIZE PORT B
0001 0920            00049         CALL     DELAY            ;DELAY 20 MSECS
0002 0915            00050         CALL     SCAN_KEYS        ;GET KEY VALUES
0003 0028            00051         MOVWF    GP               ;SAVE IN RAM
0004 0608            00052         BTFSC    GP,SW1           ;SKIP IF SW1 NOT PRESSED
0005 0929            00053         CALL     TURN_GREEN_ON    ;ELSE DO ROUTINE
0006 0628            00054         BTFSC    GP,SW2           ;SKIP IF SW2 NOT PRESSED
```

```
0007 092B          00055         CALL    TURN_RED_ON     ;ELSE DO ROUTINE
0008               00056 CHK_FOR_KEY
0008 0920          00057         CALL    DELAY           ;DELAY FOR 20 MSEC
0009 0915          00058         CALL    SCAN_KEYS       ;GET KEY HIT
000A 0F00          00059         XORLW   0               ;EXCL. OR WITH 0
000B 0743 0A08     00060         BNZ     CHK_FOR_KEY     ;KEY STILL PRESSED
                   00061                                 ;THEN LOOP
000D               00062 NO_KEY_PRESSED
000D 0446          00063         BCF     PORT_B,SCAN1    ;SET SCAN LINES LOW
000E 0466          00064         BCF     PORT_B,SCAN2    ;       /
000F 0003          00065         SLEEP                   ;SLEEP
                   00066 ;
                   00067     PAGE
                   00068 ;
0010               00069 INIT_PORT_B
0010 0C03          00070         MOVLW   B'00000011'     ; CONFIG RB0, 1 AS I/P'S
0011 0006          00071         TRIS    PORT_B          ; AND RB2-7 AS O/P'S
0012 0CFF          00072         MOVLW   0FFh
0013 0026          00073         MOVWF   PORT_B          ;DEFAULT VALUES FOR PORT_B
0014 0800          00074         RETLW   0               ;RETURN WITH NO ERROR
                   00075 ;
                   00076 ;This routine, scans two keys and returns the following:
                   00077 ;       0 if no key is pressed
                   00078 ;       1 if SW1 is pressd
                   00079 ;       2 if SW2 is pressed
                   00080 ;       3 if SW1 and SW2 are pressed
                   00081 ;
0015               00082 SCAN_KEYS
0015 0446          00083         BCF     PORT_B,SCAN1    ;ENABLE SCAN FOR SW1
0016 0466          00084         BCF     PORT_B,SCAN2    ;ENABLE SCAN FOR SW2
0017 0C03          00085         MOVLW   B'00000011'     ;LOAD MASK IN W
0018 0146          00086         ANDWF   PORT_B,0        ;AND WITH PORT
0019 0546          00087         BSF     PORT_B,SCAN1    ;DISABLE SCAN
001A 0566          00088         BSF     PORT_B,SCAN2    ;       /
001B 01E2          00089         ADDWF   PC,1            ;GET OFFSET TO TABLE
001C 0803          00090         RETLW   3               ;SW1 AND SW2 PRESSED
001D 0802          00091         RETLW   2               ;SW2 PRESSED
001E 0801          00092         RETLW   1               ;SW1 PRESSED
001F 0800          00093         RETLW   0               ;NO KEY PRESSED
                   00094 ;
                   00095 ;DELAY, IS A APPROX. WAIT FOR 20.4mSECS, FOR A SYSTEM
                   00096 ;USING A 2 Mhz CRYSTAL CLOCK.
0020               00097 DELAY
0020 0C14          00098         MOVLW   MSEC_20
0021 0028          00099         MOVWF   DB1
0022               00100 DLY1
0022 0069          00101         CLRF    DB2
0023 02E8          00102         DECFSZ  DB1, F
0024 0A26          00103         GOTO    DLY2
0025 0800          00104         RETLW   0
0026               00105 DLY2
0026 02E9          00106         DECFSZ  DB2, F          ;INNER LOOP = 1.02 MSEC.
0027 0A26          00107         GOTO    DLY2            ;       /
0028 0A22          00108         GOTO    DLY1
                   00109 ;
                   00110 ;
0029               00111 TURN_GREEN_ON
0029 0486          00112         BCF     PORT_B,GRN_LED
002A 0800          00113         RETLW   0
                   00114 ;
002B               00115 TURN_RED_ON
002B 04A6          00116         BCF     PORT_B,RED_LED
002C 0800          00117         RETLW   0
                   00118 ;
                   00119         END
```


© 1997 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXX--- ----------------

All other memory blocks unused.

Program Memory Words Used:    45
Program Memory Words Free:   467


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

AN529

Multiplexing LED Drive and a 4x4 Keypad Sampling

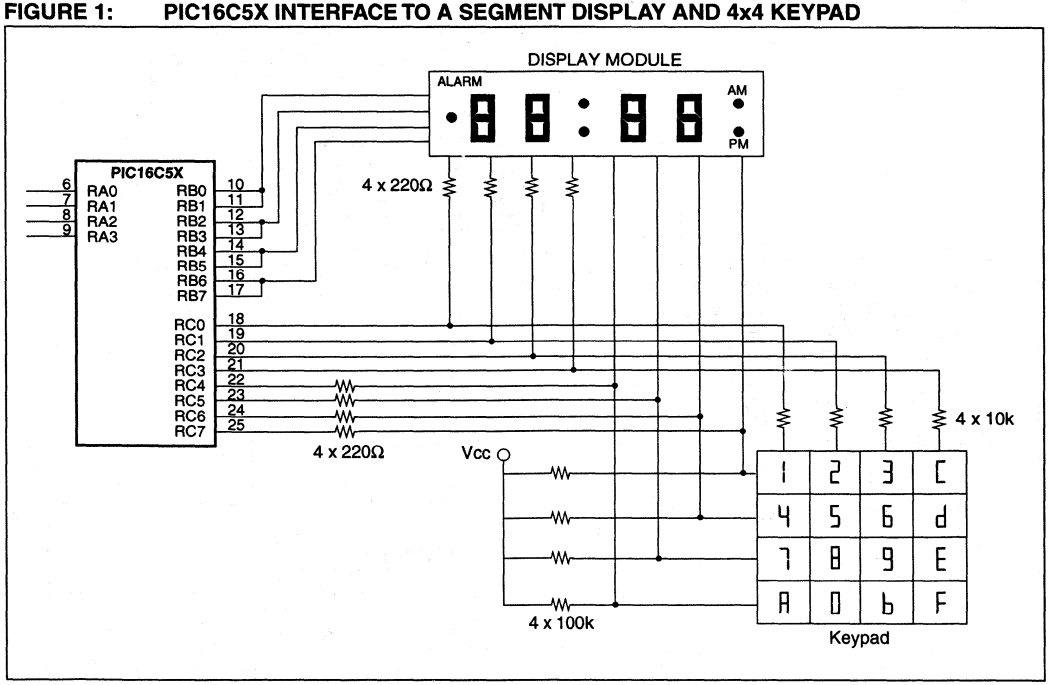
Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

Many applications require driving LEDs along with an interface to a keypad. Implementing such designs usually involves using up significant amounts of the processors I/O lines. This application note describes a method which uses only 16 I/O pins of a PIC16C5X microcontroller to sample a 4x4 keypad matrix, and directly drive four 7-segment LEDs (Figure 1). Direct drive of the LEDs is possible, because of the high sink and source capabilities of PIC16C5X microcontroller, thus eliminating the use of an external drive transistor, and results in a reduction in both cost and complexity of the overall circuit.

Typically applications having LEDs and keypads also keep track of real-time, in order to synchronize certain related events. An Industrial Clock/Timer example has been used in this application note as a demonstration of these functions. The software overhead to keep track of real time is minimal and the user can modify the code to significantly expand the functionality of this circuit.

FIGURE 1: PIC16C5X INTERFACE TO A SEGMENT DISPLAY AND 4x4 KEYPAD



© 1997 Microchip Technology Inc.

PART A: 4x4 KEY MATRIX SAMPLING

Implementation

The 4x4 Key Matrix is connected to PORTC of the PIC16C5X (Figure 2). The four columns are connected to RC0-RC3 and the four rows are connected to RC4-RC7. Each digit is refreshed every 20 ms. with a 5 ms pulse. The keypad is sampled every 20 ms with four 3 µs pulses (Figure 4).

The keypad sampling is as follows:

a) The columns are connected to output pins, and the rows are connected to input pins.

b) Each column is sequentially driven to a low voltage while at the same instance the four rows are sampled. Since the rows are all held high with pull-up resistors, all four inputs will normally be high. If a key is pressed in a column which is at a low level, that low level will be conducted to the input pin through the closed key and the corresponding row will be sensed as a low.

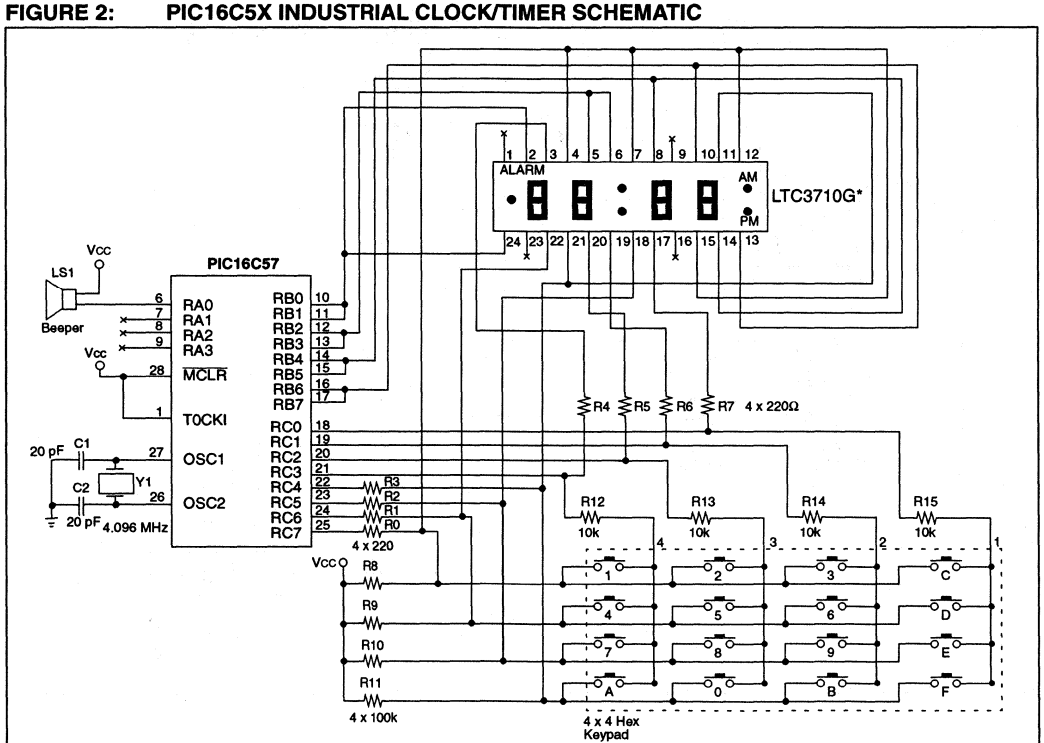
c) Before a new column is brought low, care should be taken to discharge the input pins (see code section for details).

d) A 50 ms key debounce technique has been implemented in the software, in order to eliminate multiple key strokes.

Note 1: Resistors R8-R11 and R12-R14 have been selected such that their ratio is 1:10. This will insure a 0.5V level at the input, when a key is pressed. Also R8-R14 should have a value such that their current contribution to the LEDs segments is negligible.

Note 2: In circuits where there is substantial interference between the key matrix and the LED drive circuit, the alternative circuit of Figure 3 should be utilized. Diodes in the path of all pins connected to the keypad insure that there is minimal interference from the keypad, when it is not being sampled.

FIGURE 2: PIC16C5X INDUSTRIAL CLOCK/TIMER SCHEMATIC



* Each LED segment should draw no more than 7mA.

© 1997 Microchip Technology Inc.

PART B: INDUSTRIAL CLOCK/TIMER

Clock Selection

The 4.096 MHz crystal oscillator is the time-base. The PIC16C5X internally divides the clock by 4 to give an internal clock of 1.024 MHz. This clock is further divided by 32 (by the prescaler in the OPTION register) to give a clock of 32 kHz which is used to increment the TMR0 in the PIC16C5X. If the TMR0 is initialized to 96, it would overflow to zero in 5 ms.

$$(256\text{-}96) \times (1/32000) = 5.000 \text{ ms}$$

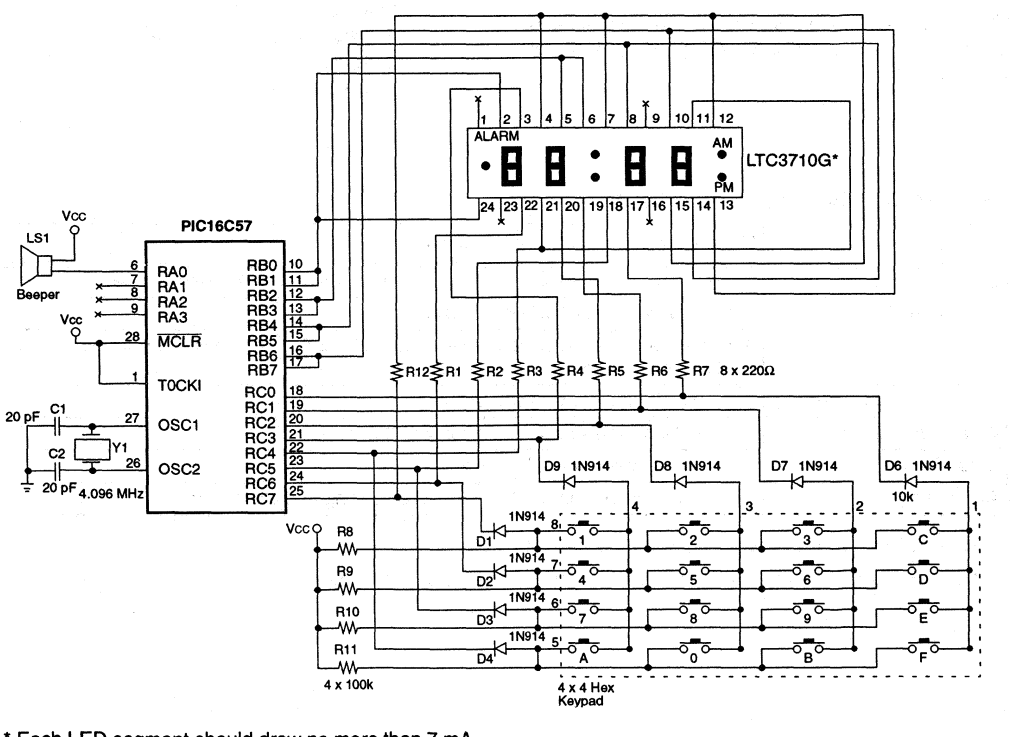
This 5 ms is used to count the seconds, minutes and hours in the clock/timer. It is also used as a time-base to update the display digits and sample the keyboard. The clock speed being 4.096 MHz, each instruction will execute in 1 µs. Therefore in 5 ms, approximately 5000 instructions can be executed. This gives sufficient time to execute a large section of code and not miss the overflow in TMR0.

Using a 3.579545 MHz color burst crystal oscillator as a time-base

Some users may want to use a color burst crystal oscillator as a time-base, because of its low cost. If a 3.579545 MHz crystal is used, then the internal clock will be 1.117 µs. If this is prescaled by 32, TMR0 will be incremented every 35.758 µs. Initializing the TMR0 with 116 will cause it to overflow to 0 in 5.006 ms. giving an error of 0.12%. This error can be corrected in software by making time adjustments every minute and/or every hour.

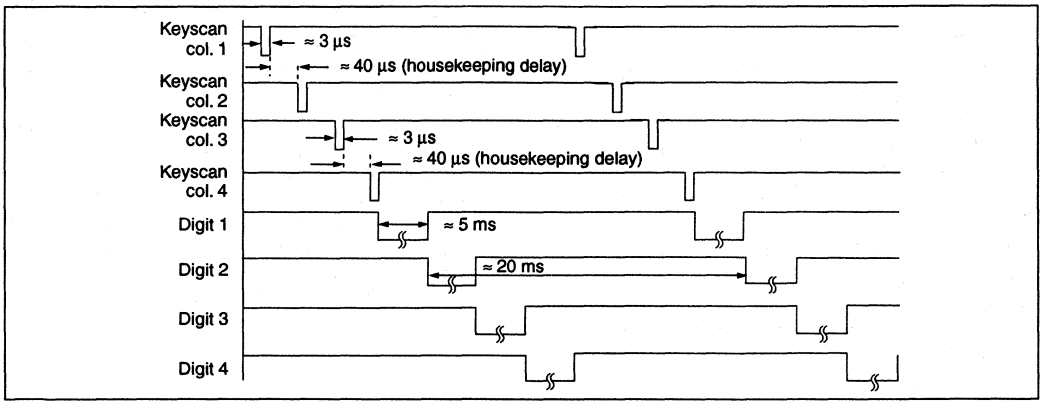
2

FIGURE 3: PIC16C5X ALARM CLOCK SCHEMATIC (USING DIODES)



* Each LED segment should draw no more than 7 mA.

FIGURE 4: KEY SCAN AND LED DIGIT SELECT TIMING



FEATURES

The Flowchart (Figure 5) shows the sequence of events in the clock/timer software. The clock has the following features:

- 12 hour clock with a.m./p.m.
- 12 hour alarm with a.m./p.m.
- Full function Hex keypad (Figure 5).
- AA audible alarm for 1 minute.
- 10 minute alarm disable.

SETTING CLOCK/TIMER FUNCTIONS

Function	Key Sequence to Activate Function
Set Real Time	Set → Hours (tens) →Hours → Minutes (tens) → Minutes → AM/PM → Set
View Alarm Time	Alarm (alarm time is displayed for 5 seconds)
Set Alarm Time	Alarm → Set (must be pressed when alarm LED is flashing) → Hours (tens) → Hours → Minutes (tens) → Minutes → AM/PM → Set
Enable/Disable Alarm	Alarm → Alarm (toggles alarm status)
Disable AA Alarm	Disable Alarm (disable audible beep for 10 minutes)
Clear Alarm	Clear Alarm (clears audible alarm)
Abort Entry	Clear Entry (aborts data entry mode when setting real alarm time)

Note 1: Valid key strokes will be acknowledged with a beep.

Note 2: Hours and minutes used above correspond to digits 0 - 9 on the keyboard.

© 1997 Microchip Technology Inc.

FIGURE 5: TIMER/CLOCK FLOWCHART

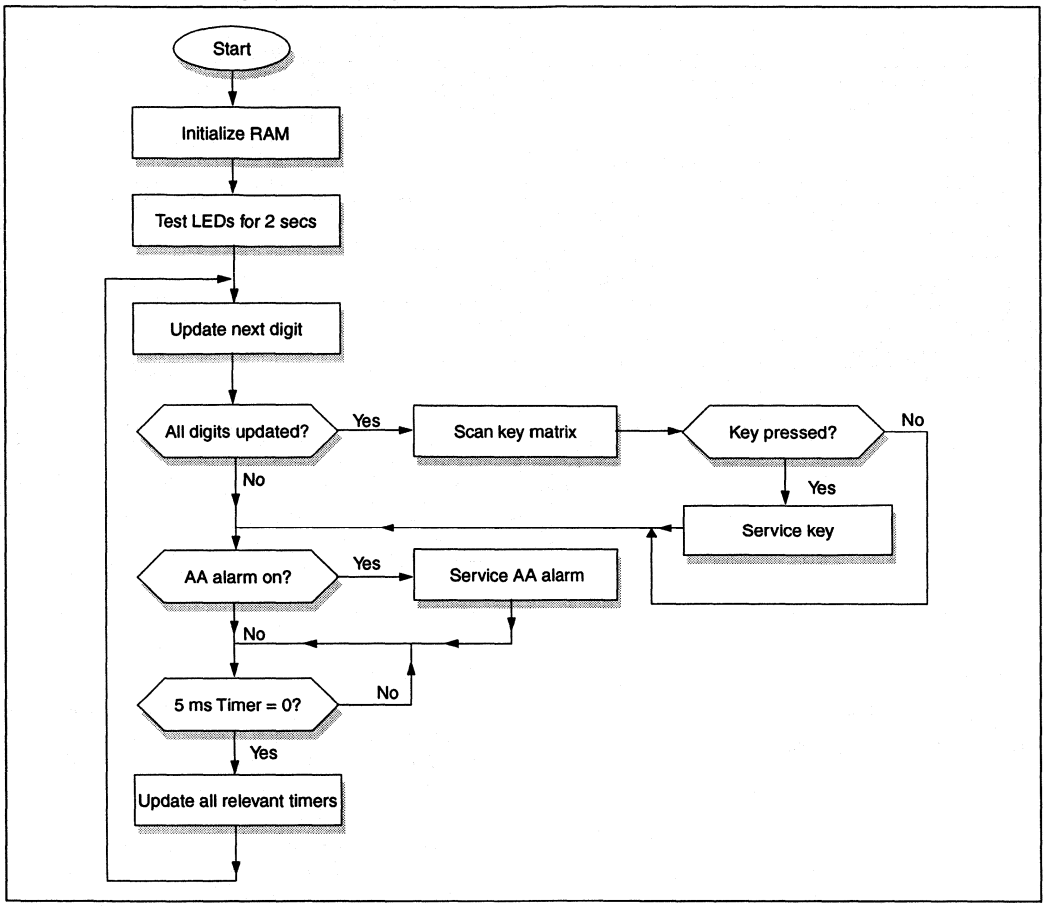
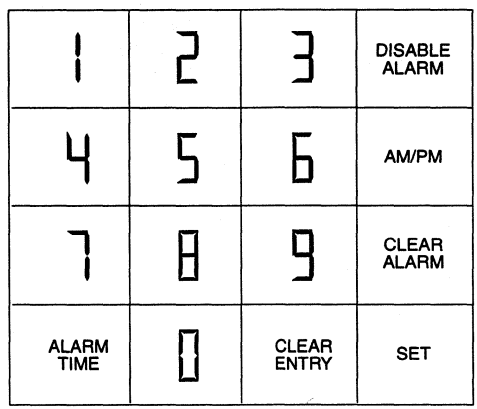
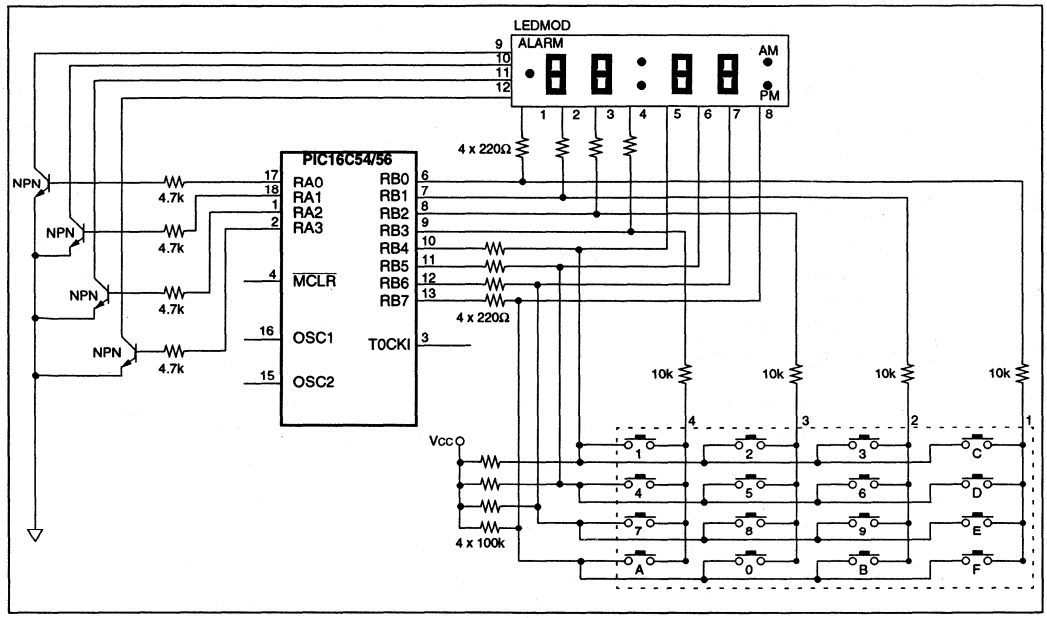


FIGURE 6: KEYPAD



© 1997 Microchip Technology Inc.

FIGURE 7: INTERFACE TO PIC16C54/56



CODE SIZE

Key scan → 97 bytes

Display update → 113 bytes

SUMMARY

This Application Note demonstrates a simple method of interfacing a PIC16C5X device to both 7-segment LEDs and a keypad. The key features of the PIC16C5X which made this possible are:

- High sink/source of the I/O ports.
- Fast instruction cycle for quick key-scan.
- RISC processor allowing minimal overhead for real-time clock maintenance.
- Reconfigurable I/O ports, enabling dual functionality of ports.

Figure 7 depicts a block diagram connecting a PIC16C54/56 to a 4-digit, 7-segment LED display and a 4x4 hex keypad. Since only 12 I/O pins are available in the PIC16C54/56, an external npn transistor will have to be utilized to sink the current from each digit.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: CODE LISTING

```
MPASM 01.40 Released              CLK.ASM   1-16-1997  13:07:37         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001                  TITLE        "Alarm Clock"
                     00002      LIST  P = 16C57
                     00003      ERRORLEVEL  -306
                     00004 ;
                     00005 ;Define Equates:
                     00006 ;
  000007FF           00007 PIC57    EQU     7FFH
                     00008 ;*****************************************************************
                     00009 ;
                     00010 ;External Ossc. used = 4.096Mhz. Prescaler of 32 used, which gives a
                     00011 ;31.25 microSec increment of the TMR0.If TMR0 is intially loaded with
                     00012 ;96, it would overflow to 0 in 5.000 milliSecs. Giving a 0.00% error.
                     00013 ;
  00000060           00014 MSEC5    EQU     D'96'
                     00015 ;
                     00016 ;
                     00017 ;        Program:          CLK.ASM
                     00018 ;        Revision Date:
                     00019 ;                          1-13-97    Compatibility with MPASMWIN 1.40
                     00020 ;
                     00021 ;*****************************************************************
                     00022 ;
  00000000           00023 C          EQU     0
  00000000           00024 BEP        EQU     0
  00000000           00025 RTATS      EQU     0
  00000001           00026 DC         EQU     1
  00000001           00027 HR10       EQU     1
  00000002           00028 Z          EQU     2
  00000002           00029 HR         EQU     2
  00000003           00030 MIN10      EQU     3
  00000004           00031 MIN        EQU     4
  00000004           00032 FLASH      EQU     4
  00000005           00033 PA0        EQU     5
  00000005           00034 KEY_BEEP   EQU     5
  00000005           00035 AMPM       EQU     5
  00000006           00036 PA1        EQU     6
  00000000           00037 F0         EQU     0
  00000006           00038 KEY_HIT    EQU     6
  00000006           00039 ALED       EQU     6
  00000007           00040 AM_PM      EQU     7
  00000003           00041 COLON      EQU     3
  00000002           00042 ALRMLED    EQU     2
  00000007           00043 SERVICED   EQU     7
  00000000           00044 ALONOF     EQU     0
  00000001           00045 INAL       EQU     1
  00000002           00046 SILNC      EQU     2
  00000003           00047 INAA       EQU     3
  00000005           00048 INKEYBEP   EQU     5
                     00049 ;
                     00050 ;DEFINE RAM LOCATIONS:
  00000001           00051 TMR0       EQU     1
  00000002           00052 PC         EQU     2
  00000003           00053 STATUS     EQU     3
  00000004           00054 FSR        EQU     4
```

```
00000005          00055 PORT_A    EQU     5
00000006          00056 PORT_B    EQU     6
00000007          00057 PORT_C    EQU     7
                  00058 ;
00000001          00059 F         EQU     1
                  00060 ;
                  00061 ;DEFINE REAL TIME MODE REGS (RTM)
00000008          00062 MSTMR     EQU     8       ;MILLI SEC. TIMER
00000009          00063 STMR      EQU     9       ;SEC. TIMER
                  00064 ;
                  00065 ;*************************************************
                  00066 ;DO NOT CHANGE RELATIVE POSITION OF NEXT 6 BYTES
0000000A          00067 MTMR      EQU     0A      ;MIN. TIMER
0000000B          00068 HTMR      EQU     0B      ;HOUR TIMER
                  00069 ;
                  00070 ;DEFINE ALARM TIME MODE REGS (ATM)
0000000C          00071 MALARM    EQU     0C      ;MIN. ALARM
0000000D          00072 HALARM    EQU     0D      ;HOUR ALARM
                  00073 ;
                  00074 ;DEFINE DATA ENTRY MODE REGS (DEM)
0000000E          00075 MENTRY    EQU     0E      ;MIN. ENTRY
0000000F          00076 HENTRY    EQU     0F      ;HOUR ENTRY
                  00077 ;*************************************************
                  00078     PAGE
                  00079 ;
                  00080 ;DEFINE FLAG REG AND FUNCTION:
00000010          00081 FLAG      EQU     10
                  00082 ;
                  00083 ;       BIT # 7|6|5|4|3|2|1|0|
                  00084 ;--------------|-|-|-|-|-|-|-|
                  00085 ;              X|X|X|X|X|X|0|0| --> REAL TIME MODE (RTM)
                  00086 ;              X|X|X|X|X|X|0|1| --> ALARM TIME MODE(ATM)
                  00087 ;              X|X|X|X|X|X|1|0| --> DATA ENTRY M0DE(DEM)
                  00088 ;              X|X|X|X|X|X|1|1| --> TEST MODE (TM)
                  00089 ;              X|X|X|X|X|Y|X|X| --> ALRMLED ON/OFF
                  00090 ;              X|X|X|X|Y|X|X|X| --> COLON LED ON/OFF
                  00091 ;              X|X|X|Y|X|X|X|X| --> FLASH DISPLAY
                  00092 ;              X|X|Y|X|X|X|X|X| --> KEY_BEEP
                  00093 ;              X|Y|X|X|X|X|X|X| --> KEY_HIT (0/1)
                  00094 ;              Y|X|X|X|X|X|X|X| --> SERVICED
                  00095 ; X = DEFINED ELSEWHERE IN TABLE
                  00096 ; Y = DEFINED AS SHOWN (0/1)
                  00097 ;
00000011          00098 TEMP      EQU     11
00000012          00099 DIGIT     EQU     12
00000013          00100 NEW_KEY   EQU     13
00000014          00101 KEY_NIBL  EQU     14
00000015          00102 DEBOUNCE  EQU     15
00000016          00103 MIN_SEC   EQU     16      ;MIN/SECONDS TIMER
00000017          00104 ENTFLG    EQU     17
                  00105 ;
                  00106 ;flag dedicated to the key entry mode
                  00107 ;       BIT # 7|6|5|4|3|2|1|0|
                  00108 ;--------------|-|-|-|-|-|-|-|
                  00109 ;              X|X|X|X|X|X|X|Y| --> REAL/ALARM TIME STATUS
                  00110 ;              X|X|X|X|X|X|Y|X| --> HR10 DONE
                  00111 ;              X|X|X|X|X|Y|X|X| --> HR DONE
                  00112 ;              X|X|X|X|Y|X|X|X| --> MIN10 DONE
                  00113 ;              X|X|X|Y|X|X|X|X| --> MIN DONE
                  00114 ;              X|X|Y|X|X|X|X|X| --> INKEYBEP
                  00115 ;              X|Y|X|Y|X|X|X|X| --> NOT USED
                  00116 ;              Y|X|X|X|X|X|X|X| --> NOT USED
                  00117 ;
                  00118 ;
00000018          00119 ALFLAG    EQU     18
                  00120 ;
```


© 1997 Microchip Technology Inc.

```
                    00121 ;flag dedicated to the alarm
                    00122 ;       BIT # 7|6|5|4|3|2|1|0|
                    00123 ;--------------|-|-|-|-|-|-|-|
                    00124 ;              X|X|X|X|X|X|X|Y| --> ALONOF
                    00125 ;              X|X|X|X|X|X|Y|X| --> INAL
                    00126 ;              X|X|X|X|X|Y|X|X| --> SILNC
                    00127 ;              X|X|X|X|Y|X|X|X| --> INAA
                    00128 ;              X|X|X|Y|X|X|X|X| --> NOT USED
                    00129 ;              X|X|Y|X|X|X|X|X| --> NOT USED
                    00130 ;              X|Y|X|Y|X|X|X|X| --> NOT USED
                    00131 ;              Y|X|X|X|X|X|X|X| --> NOT USED
                    00132 ;
00000019            00133 AAFLAG    EQU     19
                    00134 ;
                    00135 ;flag dedicated to the AA alarm
0000001A            00136 AATMR     EQU     1A
                    00137 ;
                    00138 ;Port pin definitions:
                    00139 ;
                    00140 ;PORT_A:
                    00141 ;       BIT 0   --> BEEPER (ACTIVE LOW) OUTPUT
                    00142 ;       BIT 1-3 --> unused I/O
                    00143 ;
                    00144 ;PORT_B: ALL OUTPUTS
                    00145 ;       BIT 0&4 --> MSB DIGIT COMMON CATHODE & ALARM
                    00146 ;       BIT 1&5 --> 2ND DIGIT COMMOM CATHODE & COLON
                    00147 ;       BIT 2&6 --> 3RD DIGIT COMMON CATHODE & PM
                    00148 ;       BIT 3&7 --> LSB DIGIT COMMON CATHODE & AM
                    00149 ;
                    00150 ;PORT_C:
                    00151 ;IN DISPLAY MODE ALL SEG/ANNN SET AS OUTPUTS
                    00152 ;IN KEY SCAN MODE COLS ARE OUTPUTS ROWS ARE INPUTS
                    00153 ;       BIT 0   --> SEGMENT A & COL 4
                    00154 ;       BIT 1   --> SEGMENT B & COL 3
                    00155 ;       BIT 2   --> SEGMENT C & COL 2
                    00156 ;       BIT 3   --> SEGMENT D & COL 1
                    00157 ;       BIT 4   --> SEGMENT E & ROW 4
                    00158 ;       BIT 5   --> SEGMENT F & ROW 3
                    00159 ;       BIT 6   --> SEGMENT G & ROW 2
                    00160 ;       BIT 7   --> CA OF ALL ANNUNCIATORS & ROW 1
                    00161 ;
                    00162 ;
                    00163     PAGE
                    00164 ;
0000                00165         ORG     0
0000                00166 START
0000 0AFC           00167         GOTO    INIT_CLK        ;INITIALIZE CLOCK
                    00168 ;THIS ROUTINE RUNS A TEST ON THE LEDS.
                    00169 ;ALL THE RELEVENT LEDS ARE LIT UP FOR 2 SECS.
                    00170 ;
0001                00171 TEST_HARDWARE
0001 0C02           00172         MOVLW   d'02'           ;DISPLAY FOR 2 SECS
0002 0036           00173         MOVWF   MIN_SEC         ;       /
                    00174 ;
                    00175 ;
0003                00176 TEST_LOOP
0003 0216           00177         MOVF    MIN_SEC,W       ;GET MIN/SEC
0004 0643           00178         BTFSC   STATUS,Z        ;NOT 0 THEN SKIP
0005 0A0B           00179         GOTO    NORM_TIME       ;ELSE NORMAL TIME
0006 0925           00180         CALL    UPDATE_DISPLAY  ;UPDATE DISPLAY
0007 05A3           00181         BSF     STATUS,PA0      ;GOTO PAGE 1
0008 0900           00182         CALL    UPDATE_TIMERS   ;WAIT AND UPDATE
0009 04A3           00183         BCF     STATUS,PA0      ;RESET PAGE MARKER
000A 0A03           00184         GOTO    TEST_LOOP       ;LOOP BACK
000B                00185 NORM_TIME
000B 0410           00186         BCF     FLAG,0          ;PUT IN REAL TIME
```

```
000C 0430           00187         BCF     FLAG,1
000D                00188 TIME_LOOP
000D 0925           00189         CALL    UPDATE_DISPLAY
000E 05C3           00190         BSF     STATUS,PA1      ;GOTO PAGE 2
000F 0900           00191         CALL    SERVICE_KEYS
0010 05A3           00192         BSF     STATUS,PA0      ;GOTO PAGE 3
0011 0900           00193         CALL    SOUND_AA        ;CHECK ALARM
0012 04C3           00194         BCF     STATUS,PA1      ;GOTO PAGE 1
0013 0900           00195         CALL    UPDATE_TIMERS   ;WAIT AND UPDATE TIMERS
0014 04A3           00196         BCF     STATUS,PA0      ;RESET PAGE MARKER
0015 04C3           00197         BCF     STATUS,PA1      ;       /
0016 0210           00198         MOVF    FLAG,W          ;SEE IF IN ATM
0017 0E03           00199         ANDLW   B'00000011'     ;       /
0018 0F01           00200         XORLW   B'00000001'     ;      /
0019 0643           00201         BTFSC   STATUS,Z        ;SKIP IF NOT
001A 091C           00202         CALL    RESET_ATM
001B 0A0D           00203         GOTO    TIME_LOOP
                    00204 ;
001C                00205 RESET_ATM
001C 0216           00206         MOVF    MIN_SEC,W       ;GET MIN/SEC
001D 0E0F           00207         ANDLW   B'00001111'     ;       /
001E 0743           00208         BTFSS   STATUS,Z        ;Z THEN SKIP
001F 0800           00209         RETLW   0               ;ELSE RETURN
0020 0410           00210         BCF     FLAG,0          ;SET TO RTM
0021 0450           00211         BCF     FLAG,ALRMLED    ;CLEAR LED
0022 0618           00212         BTFSC   ALFLAG,ALONOF   ;TEST STAT
0023 0550           00213         BSF     FLAG,ALRMLED    ;SET LED
0024 0800           00214         RETLW   0               ;RETURN
                    00215     PAGE
                    00216 ;
                    00217 ;
0025                00218 UPDATE_DISPLAY
0025 0C00           00219         MOVLW   B'00000000'     ;CLEAR SEG DRIVE
0026 0027           00220         MOVWF   PORT_C          ;       /
0027 0C3F           00221         MOVLW   B'00111111'     ;SEE IF LAST DIGIT
0028 0186           00222         XORWF   PORT_B,0        ;       /
0029 0643           00223         BTFSC   STATUS,Z        ;NO THEN SKIP
002A 0A6F           00224         GOTO    SCAN_KP         ;ELSE SCAN KEYPAD
002B                00225 UP_DSP_1
                    00226 ;SELECT DIGIT TO BE DISPLAYED
002B 0246           00227         COMF    PORT_B,0        ;GET COMPL. PORT B IN W
002C 0643           00228         BTFSC   STATUS,Z        ;NO DIGIT SELECTED?
002D 0CC0           00229         MOVLW   B'11000000'     ;THEN SELECT DEFAULT
002E 0031           00230         MOVWF   TEMP            ;SAVE IN TEMP
002F 0271           00231         COMF    TEMP, F         ;COMPLEMENT VALUE
0030 0503           00232         BSF     STATUS,C        ;SET CARRY
0031 0371           00233         RLF     TEMP, F         ;SHIFT LEFT
0032 0703           00234         BTFSS   STATUS,C        ;IF C=1 THEN SKIP
0033 0371           00235         RLF     TEMP, F         ;ELSE 3 TIMES...
0034 0371           00236         RLF     TEMP, F         ;THRU CARRY
0035 0211           00237         MOVF    TEMP,0          ;GET IN W
0036 0026           00238         MOVWF   PORT_B          ;OUTPUT TO PORT
                    00239 ;NOW THAT DIGIT IS SELECTED, SELECT SEG VALUES FOR THAT DIGIT
                    00240 ;FIRST FIND MODE OF OPERATION.
0037 0C0A           00241         MOVLW   MTMR            ;LOAD FSR WITH MTMR
0038 0024           00242         MOVWF   FSR             ;       /
0039 0210           00243         MOVF    FLAG,0          ;GET FLAG IN W
003A 0E03           00244         ANDLW   B'00000011'     ;MASK OTHER BITS
003B 0031           00245         MOVWF   TEMP            ;SAVE IN TEMP
003C 0F03           00246         XORLW   B'00000011'     ;IN TEST MODE
003D 0643           00247         BTFSC   STATUS,Z        ;NO THEN SKIP
003E 0A4B           00248         GOTO    DO_TM           ;ELSE TEST MODE
003F 0403           00249         BCF     STATUS,C        ;CLEAR CARRY
0040 0371           00250         RLF     TEMP, F         ;LEFT SHIFT TEMP
0041 0211           00251         MOVF    TEMP,0          ;GET IN W
0042 01E4           00252         ADDWF   FSR, F          ;CHANGE INDIRECT POINTER
```


© 1997 Microchip Technology Inc.

```
0043 0954          00253          CALL    GET_7_SEG       ;GET 7 SEG DATA IN W
0044 0032          00254          MOVWF   DIGIT           ;SAVE IN DIGIT LOC.
0045 09D1          00255          CALL    MASK_ANNC       ;MASK ANNC TO DIGIT
0046 0690          00256          BTFSC   FLAG,FLASH      ;NO FLASH THEN SKIP
0047 094E          00257          CALL    CHK_HALF_SEC    ;ELSE CHK. IF ON
0048 0212          00258          MOVF    DIGIT,0         ;GET BACK DIGIT
0049 0027          00259          MOVWF   PORT_C          ;OUTPUT TO PORT
004A 0800          00260          RETLW   0               ;RETURN
                   00261 ;
004B               00262 DO_TM
004B 0CFF          00263          MOVLW   B'11111111'     ;LIGHT ALL SEGMENTS
004C 0027          00264          MOVWF   PORT_C          ;       /
004D 0800          00265          RETLW   0               ;RETURN FROM UPDATE DISPLAY
                   00266 ;
004E               00267 CHK_HALF_SEC
004E 0770          00268          BTFSS   FLAG,COLON      ;IF COLON ON THEN DO
004F 0A51          00269          GOTO    BLANK_DSP       ;ELSE BLANK DISPLAY
0050 0800          00270          RETLW   0
0051               00271 BLANK_DSP
0051 0C00          00272          MOVLW   B'00000000'     ;MAKE PORT C LOW
0052 0032          00273          MOVWF   DIGIT
0053 0800          00274          RETLW   0
                   00275 ;
                   00276      PAGE
                   00277 ;
                   00278 ;ON ENTRY FSR POINTS TO THE REAL TIME MODE'S MINUTES REGISTER.
                   00279 ;ON RETURN FSR POINTS TO THE TIMER REGISTER TO BE DISPLAYED.
                   00280 ;W REG. CONTAINS THE DECODED 7 SEG. INFO OF THE DIGIT
                   00281 ;TO BE DISPLAYED
                   00282 ;
0054               00283 GET_7_SEG
0054 0246          00284          COMF    PORT_B,0        ;COMPLEMENT B --> W
0055 0EF0          00285          ANDLW   B'11110000'     ;MASK LO NIBBLE
0056 0643          00286          BTFSC   STATUS,Z        ;NZ THEN SKIP
0057 02A4          00287          INCF    FSR, F          ;INC POINTER
0058 0200          00288          MOVF    F0,0            ;MOVE INDIRECT TO W
0059 0031          00289          MOVWF   TEMP            ;GET INTO TEMP
005A 0246          00290          COMF    PORT_B,0        ;COMPL.B --> W
005B 0EF0          00291          ANDLW   B'11110000'     ;MASK LO NIBBLE
005C 0643          00292          BTFSC   STATUS,Z        ;IF D1/2 THEN
005D 04F1          00293          BCF     TEMP,AM_PM      ;CLEAR AM/PM BIT
005E 0246          00294          COMF    PORT_B,0        ;GET PORT B AGAIN
005F 0ECC          00295          ANDLW   B'11001100'     ;SEE IF D2 OR D4
0060 0643          00296          BTFSC   STATUS,Z        ;YES THEN SKIP
0061 03B1          00297          SWAPF   TEMP, F         ;SWAP TEMP
0062 0C0F          00298          MOVLW   B'00001111'     ;MASK HI NIBBLE
0063 0151          00299          ANDWF   TEMP,0
0064 01E2          00300          ADDWF   PC, F           ;ADD TO PC
0065 083F          00301          RETLW   B'00111111'     ;CODE FOR 0
0066 0806          00302          RETLW   B'00000110'     ;CODE FOR 1
0067 085B          00303          RETLW   B'01011011'     ;CODE FOR 2
0068 084F          00304          RETLW   B'01001111'     ;CODE FOR 3
0069 0866          00305          RETLW   B'01100110'     ;CODE FOR 4
006A 086D          00306          RETLW   B'01101101'     ;CODE FOR 5
006B 087D          00307          RETLW   B'01111101'     ;CODE FOR 6
006C 0807          00308          RETLW   B'00000111'     ;CODE FOR 7
006D 087F          00309          RETLW   B'01111111'     ;CODE FOR 8
006E 0867          00310          RETLW   B'01100111'     ;CODE FOR 9
                   00311 ;
                   00312 ;This routine scans the 4x4 hex key pad for a key hit.
                   00313 ;If key is pressed, KEY_HIT flag is set and the value of
                   00314 ;the hex key is returned in reg NEW_KEY
                   00315 ;If no key is detected, then a 0xff value is returned in
                   00316 ;register NEW_KEY and the flag KEY_HIT is reset.
                   00317 ;
006F               00318 SCAN_KP
```

```
006F 06D0          00319          BTFSC   FLAG,KEY_HIT    ;KEY UNDER SERVICE?
0070 0A2B          00320          GOTO    UP_DSP_1        ;YES SKIP ROUTINE
0071 0CFF          00321          MOVLW   B'11111111'     ;SET DIGIT SINKS ...
0072 0026          00322          MOVWF   PORT_B          ;TO HIGH
0073 0CF7          00323          MOVLW   B'11110111'     ;SET KEY COL LOW
0074 0031          00324          MOVWF   TEMP            ;SAVE IN TEMP
0075               00325 SKP1
0075 0C00          00326          MOVLW   B'00000000'     ;SET PORT C AS OUTPUTS
0076 0007          00327          TRIS    PORT_C          ;       /
0077 0211          00328          MOVF    TEMP,W
0078 0E0F          00329          ANDLW   B'00001111'     ;DISCHARGE PINS FOR MEMBRANE KEYPADS
0079 0027          00330          MOVWF   PORT_C          ;       /
007A 0CF0          00331          MOVLW   B'11110000'     ;SET AS I/O
007B 0007          00332          TRIS    PORT_C          ;   /
007C 0211          00333          MOVF    TEMP,W          ;GET OLD VALUE
007D 0027          00334          MOVWF   PORT_C          ;OUTPUT TO PORT
007E 0207          00335          MOVF    PORT_C,W        ;INPUT PORT VALUE
007F 0EF0          00336          ANDLW   B'11110000'     ;MASK LO BYTE
0080 0FF0          00337          XORLW   B'11110000'     ;SEE IF KEY HIT
0081 0743          00338          BTFSS   STATUS,Z        ;NO KEY THEN SKIP
0082 0A8D          00339          GOTO    DET_KEY         ;LOAD KEY VALUE
0083               00340 SKP3
0083 0503          00341          BSF     STATUS,C        ;SET CARRY
0084 0331          00342          RRF     TEMP, F         ;MAKE NEXT COL. LOW
0085 0603          00343          BTFSC   STATUS,C        ;ALL DONE THEN SKIP
0086 0A75          00344          GOTO    SKP1
0087 0073          00345          CLRF    NEW_KEY         ;SET NEW_KEY = FF
0088 00F3          00346          DECF    NEW_KEY, F      ;       /
0089               00347 SKP2
0089 0067          00348          CLRF    PORT_C          ;SETPORT C AS ...
008A 0C00          00349          MOVLW   B'00000000'     ;OUTPUTS
008B 0007          00350          TRIS    PORT_C          ;       /
008C 0A2B          00351          GOTO    UP_DSP_1        ;RETURN
008D               00352 DET_KEY
                   00353 ;key is detected
008D 0293          00354          INCF    NEW_KEY,W       ;CHK IF KEY ...
008E 0743          00355          BTFSS   STATUS,Z        ;WAS RELEASED
008F 0A89          00356          GOTO    SKP2            ;NO THEN RETURN
0090 0207          00357          MOVF    PORT_C,W        ;GET RAW KEY...
0091 0D0F          00358          IORLW   B'00001111'     ;VALUE.
0092 0151          00359          ANDWF   TEMP,W          ;   /
0093 0033          00360          MOVWF   NEW_KEY         ;SAVE IN NEW_KEY
0094 0998          00361          CALL    GET_KEY_VAL     ;GET ACTUAL KEY ...
0095 0033          00362          MOVWF   NEW_KEY         ;VALUE
0096 05D0          00363          BSF     FLAG,KEY_HIT    ;SET KEY HIT FLAG
0097 0A89          00364          GOTO    SKP2            ;RETURN
                   00365 ;
                   00366      PAGE
                   00367 ;This routine decodes the hex value from the "raw" data got
                   00368 ;from scanning the rows and cols.
                   00369 ;   actual key value          raw hex value
                   00370 ;       ONE     EQU     77
                   00371 ;       TWO     EQU     7B
                   00372 ;       THREE   EQU     7D
                   00373 ;       C       EQU     7E
                   00374 ;       FOUR    EQU     0B7
                   00375 ;       FIVE    EQU     0BB
                   00376 ;       SIX     EQU     0BD
                   00377 ;       D       EQU     0BE
                   00378 ;       SEVEN   EQU     0D7
                   00379 ;       EIGHT   EQU     0DB
                   00380 ;       NINE    EQU     0DD
                   00381 ;       E       EQU     0DE
                   00382 ;       A       EQU     0E7
                   00383 ;       ZERO    EQU     0EB
                   00384 ;       B       EQU     0ED
```


© 1997 Microchip Technology Inc.

```
                    00385 ;       F       EQU     0EE
                    00386 ;
                    00387 ;
0098                00388 GET_KEY_VAL
0098 0E0F           00389         ANDLW   B'00001111'     ;GET LO NIBBLE
0099 0034           00390         MOVWF   KEY_NIBL        ;SAVE
009A 0C04           00391         MOVLW   4               ;SET COUNT TO 4
009B 0031           00392         MOVWF   TEMP            ;       /
009C                00393 GKV1
009C 0503           00394         BSF     STATUS,C        ;SET CARRY
009D 0334           00395         RRF     KEY_NIBL, F     ;ROTATE NIBBLE
009E 0703           00396         BTFSS   STATUS,C        ;SKIP IF NOT Z
009F 0AA5           00397         GOTO    GET_HI_KEY      ;GOTO NEXT PART
00A0 02F1           00398         DECFSZ  TEMP, F         ;DEC COUNT
00A1 0A9C           00399         GOTO    GKV1            ;LOOP
00A2                00400 GO_RESET
00A2 05A3           00401         BSF     STATUS,PA0      ;SET MSB
00A3 05C3           00402         BSF     STATUS,PA1      ;       /
00A4 0BFF           00403         GOTO    SYS_RESET       ;ELSE BIG ERROR
00A5                00404 GET_HI_KEY
00A5 00F1           00405         DECF    TEMP, F         ;REDUCE BY 1
00A6 0393           00406         SWAPF   NEW_KEY,W       ;GET HI NIBBLE
00A7 0E0F           00407         ANDLW   B'00001111'     ;       /
00A8 0034           00408         MOVWF   KEY_NIBL        ;SAVE
00A9 0211           00409         MOVF    TEMP,W          ;GET OFFSET TO TBL
00AA 01E2           00410         ADDWF   PC, F           ;LOAD IN PC
00AB 0AAF           00411         GOTO    GET147A         ;JUMP TO NEXT PART
00AC 0AB8           00412         GOTO    GET2580         ;       /
00AD 0ABA           00413         GOTO    GET369B         ;     /
00AE 0ABC           00414         GOTO    GETCDEF         ;   /
                    00415 ;
00AF                00416 GET147A
00AF 0C04           00417         MOVLW   4               ;SET COUNT TO 4
00B0                00418 GETCOM
00B0 0031           00419         MOVWF   TEMP            ;
00B1                00420 GETCOM1
00B1 0503           00421         BSF     STATUS,C        ;SET CARRY
00B2 0334           00422         RRF     KEY_NIBL, F     ;ROTATE RIGHT
00B3 0703           00423         BTFSS   STATUS,C        ;CHECK IF DONE
00B4 0ABE           00424         GOTO    KEY_TBL         ;JUMP TO TABLE
00B5 02F1           00425         DECFSZ  TEMP, F         ;DEC COUNT
00B6 0AB1           00426         GOTO    GETCOM1         ;LOOP
00B7 0AA2           00427         GOTO    GO_RESET        ;ELSE ERROR
                    00428 ;
00B8                00429 GET2580
00B8 0C08           00430         MOVLW   8               ;SET COUNT TO 8
00B9 0AB0           00431         GOTO    GETCOM
                    00432 ;
00BA                00433 GET369B
00BA 0C0C           00434         MOVLW   D'12'           ;SET COUNT TO 12
00BB 0AB0           00435         GOTO    GETCOM
                    00436 ;
00BC                00437 GETCDEF
00BC 0C10           00438         MOVLW   D'16'           ;SET COUNT TO 16
00BD 0AB0           00439         GOTO    GETCOM
                    00440     PAGE
                    00441 ;
00BE                00442 KEY_TBL
00BE 00F1           00443         DECF    TEMP, F         ;REDUCE BY 1
00BF 0211           00444         MOVF    TEMP,W          ;GET IN W
00C0 01E2           00445         ADDWF   PC, F           ;JUMP TO TABLE
00C1 0801           00446         RETLW   1               ;KEY 1
00C2 0804           00447         RETLW   4               ;KEY 4
00C3 0807           00448         RETLW   7               ;KEY 7
00C4 080A           00449         RETLW   0A              ;KEY A
00C5 0802           00450         RETLW   2               ;KEY 2
```


© 1997 Microchip Technology Inc.

```
00C6 0805          00451          RETLW   5               ;KEY 5
00C7 0808          00452          RETLW   8               ;KEY 8
00C8 0800          00453          RETLW   0               ;KEY 0
00C9 0803          00454          RETLW   3               ;KEY 3
00CA 0806          00455          RETLW   6               ;KEY 6
00CB 0809          00456          RETLW   9               ;KEY 9
00CC 080B          00457          RETLW   0B              ;KEY B
00CD 080C          00458          RETLW   0C              ;KEY C
00CE 080D          00459          RETLW   0D              ;KEY D
00CF 080E          00460          RETLW   0E              ;KEY E
00D0 080F          00461          RETLW   0F              ;KEY F
                   00462 ;
                   00463      PAGE
                   00464 ;
00D1               00465 MASK_ANNC
00D1 0CFC          00466          MOVLW   B'11111100'     ;CHK IF DIGIT 1
00D2 0186          00467          XORWF   PORT_B,0        ;       /
00D3 0643          00468          BTFSC   STATUS,Z        ;NO THEN SKIP
00D4 0AE5          00469          GOTO    MASK_ALARM      ;ELSE MASK ALARM
00D5 0CF3          00470          MOVLW   B'11110011'     ;CHK IF DIGIT 2
00D6 0186          00471          XORWF   PORT_B,0        ;       /
00D7 0643          00472          BTFSC   STATUS,Z        ;NO THEN SKIP
00D8 0AE8          00473          GOTO    MASK_COLON      ;ELSE MASK COLON
00D9 0CCF          00474          MOVLW   B'11001111'     ;CHK IF DIGIT 3
00DA 0186          00475          XORWF   PORT_B,0        ;       /
00DB 0643          00476          BTFSC   STATUS,Z        ;NO THEN SKIP
00DC 0AE1          00477          GOTO    MASK_PM         ;ELSE MASK PM
00DD               00478 MASK_AM
00DD 02A4          00479          INCF    FSR, F          ;INC FSR
00DE 07E0          00480          BTFSS   F0,AM_PM        ;IF 0 THEN AM
00DF 05F2          00481          BSF     DIGIT,7         ;SET MSB
00E0 0AEB          00482          GOTO    BLNK_LEAD_0     ;NEXT
00E1               00483 MASK_PM
00E1 02A4          00484          INCF    FSR, F          ;INC FSR
00E2 06E0          00485          BTFSC   F0,AM_PM        ;IF 1 THEN PM
00E3 05F2          00486          BSF     DIGIT,7         ;SET MSB
00E4 0AEB          00487          GOTO    BLNK_LEAD_0     ;NEXT
00E5               00488 MASK_ALARM
00E5 0650          00489          BTFSC   FLAG,ALRMLED    ;1 THEN LIGHT LED
00E6 05F2          00490          BSF     DIGIT,7         ;       /
00E7 0AEB          00491          GOTO    BLNK_LEAD_0     ;NEXT
00E8               00492 MASK_COLON
00E8 0670          00493          BTFSC   FLAG,COLON      ;1 THEN LIGHT LED
00E9 05F2          00494          BSF     DIGIT,7         ;       /
00EA 0AEB          00495          GOTO    BLNK_LEAD_0     ;NEXT
                   00496 ;
00EB               00497 BLNK_LEAD_0
00EB 0210          00498          MOVF    FLAG,W          ;GET IN W
00EC 0E03          00499          ANDLW   B'00000011'     ;SEE IF IN DEM
00ED 0F02          00500          XORLW   B'00000010'     ;CHECK
00EE 0643          00501          BTFSC   STATUS,Z        ;NO THEN DO
00EF 0800          00502          RETLW   0               ;ELSE RETURN
00F0 0CFC          00503          MOVLW   B'11111100'     ;SEE IF DIGIT 1
00F1 0186          00504          XORWF   PORT_B,0        ;       /
00F2 0743          00505          BTFSS   STATUS,Z        ;YES THEN SKIP
00F3 0800          00506          RETLW   0               ;RETURN
00F4 0C3F          00507          MOVLW   B'00111111'     ;ELSE MASK G AND ANUNC
00F5 0152          00508          ANDWF   DIGIT,0         ;GET IN W
00F6 0F3F          00509          XORLW   B'00111111'     ;SEE IF 0
00F7 0743          00510          BTFSS   STATUS,Z        ;YES THEN SKIP
00F8 0800          00511          RETLW   0               ;RETURN
00F9 0C80          00512          MOVLW   B'10000000'     ;ELSE BLANK D1
00FA 0172          00513          ANDWF   DIGIT, F        ;       /
00FB 0800          00514          RETLW   0               ;RETURN
                   00515 ;
                   00516 ;
```


© 1997 Microchip Technology Inc.

```
                    00517 ;
                    00518     PAGE
                    00519 ;
                    00520 ;THIS ROUTINE SETS UP PORTS A,B,C AND THE INTERNAL
                    00521 ;REAL TIME CLOCK COUNTER.
00FC                00522 INIT_CLK
00FC 0C0F           00523         MOVLW   B'00001111'     ;MAKE ACTIVE HIGH
00FD 0025           00524         MOVWF   PORT_A          ;       /
00FE 0C00           00525         MOVLW   B'00000000'     ;SET PORT A AS OUTPUTS
00FF 0005           00526         TRIS    PORT_A
                    00527 ;
0100 0CFF           00528         MOVLW   B'11111111'     ;SET LEVELS HIGH
0101 0026           00529         MOVWF   PORT_B          ;       /
0102 0C00           00530         MOVLW   B'00000000'     ;SET PORT B AS OUTPUTS
0103 0006           00531         TRIS    PORT_B
                    00532 ;
0104 0C00           00533         MOVLW   B'00000000'     ;SET LEVELS LOW
0105 0027           00534         MOVWF   PORT_C          ;       /
0106 0C00           00535         MOVLW   B'00000000'     ;SET PORT C AS OUTPUTS
0107 0007           00536         TRIS    PORT_C          ;       /
                    00537 ;
0108 0C04           00538         MOVLW   B'00000100'     ;SET UP PRESCALER
0109 0002           00539         OPTION                  ;       /
                    00540 ;
010A 0C60           00541         MOVLW   MSEC5           ;TMR0 = 5 mSEC
010B 0021           00542         MOVWF   TMR0            ;       /
010C 0068           00543         CLRF    MSTMR           ;CLEAR MSTMR
010D 0069           00544         CLRF    STMR            ; & SEC TMR
010E 006A           00545         CLRF    MTMR            ;& MINUTES
010F 0C12           00546         MOVLW   12H             ;MAKE HRS = 12
0110 002B           00547         MOVWF   HTMR            ;     /
0111 002D           00548         MOVWF   HALARM          ;MAKE HRS = 12
0112 006C           00549         CLRF    MALARM          ;       /
0113 0C03           00550         MOVLW   B'00000011'     ;SET TO TEST MODE
0114 0030           00551         MOVWF   FLAG            ;       /
0115 0078           00552         CLRF    ALFLAG          ;CLEAR ALL FLAG
0116 0079           00553         CLRF    AAFLAG          ;       /
0117 0077           00554         CLRF    ENTFLG          ;      /
0118 0A01           00555         GOTO    TEST_HARDWARE
                    00556 ;
                    00557 ;All routines related to timer updates are located at
                    00558 ;address 200 and above.
                    00559     PAGE
0200                00560         ORG     0200
                    00561 ;
0200                00562 UPDATE_TIMERS
0200 0201           00563         MOVF    TMR0,W          ;SEE IF TMR0 = 0
0201 0743           00564         BTFSS   STATUS,Z        ;IF 0 THEN SKIP
0202 0A00           00565         GOTO    UPDATE_TIMERS   ;ELSE LOOP
0203 0C60           00566         MOVLW   MSEC5           ;TMR0 = 5 mSEC
0204 0021           00567         MOVWF   TMR0            ;       /
0205 02A8           00568         INCF    MSTMR, F        ;INC 5 MILLI SEC
0206 06D0           00569         BTFSC   FLAG,KEY_HIT    ;NO KEY HIT THEN SKIP
0207 0A70           00570         GOTO    CHK_DE_BOUNCE   ;ELSE DEBOUNCE
0208                00571 UP_TMR_1
0208 0210           00572         MOVF    FLAG,W          ;ALARM MODE?
0209 0E03           00573         ANDLW   B'00000011'     ;       /
020A 0F01           00574         XORLW   B'00000001'     ;       /
020B 0743           00575         BTFSS   STATUS,Z        ;SKIP IF YES
020C 0A14           00576         GOTO    UP_TMR_2        ;DO NEXT
020D 0550           00577         BSF     FLAG,ALRMLED    ;LIGHT LED
020E 0570           00578         BSF     FLAG,COLON      ;       /
020F 0C64           00579         MOVLW   D'100'          ;IF 1/2 SEC
0210 0088           00580         SUBWF   MSTMR,0         ; BLINK
0211 0703           00581         BTFSS   STATUS,C        ;  /
0212 0450           00582         BCF     FLAG,ALRMLED    ;ALARM LED
```

```
0213 0A19           00583          GOTO    UP_TMR_3         ;SKIP
0214                00584 UP_TMR_2
0214 0570           00585          BSF     FLAG,COLON       ;TURN ON
0215 0C64           00586          MOVLW   D'100'           ;<100 BLINK COLON
0216 0088           00587          SUBWF   MSTMR,0          ;       /
0217 0703           00588          BTFSS   STATUS,C         ;YES THEN SKIP
0218 0470           00589          BCF     FLAG,COLON       ;ELSE TURN OFF
0219                00590 UP_TMR_3
0219 0208           00591          MOVF    MSTMR,0          ;GET MSTMR IN W
021A 0FC7           00592          XORLW   D'199'           ;= 199 THEN SKIP
021B 0743           00593          BTFSS   STATUS,Z         ;       /
021C 0800           00594          RETLW   0
                    00595 ;INC SECONDS COUNT
021D 0068           00596          CLRF    MSTMR            ;CLEAR MS_TMR
021E 0216           00597          MOVF    MIN_SEC,W        ;GET MIN_SEC TIMER
021F 0E0F           00598          ANDLW   B'00001111'      ;MASK MINUTES
0220 0743           00599          BTFSS   STATUS,Z         ;ZERO THEN SKIP
0221 00F6           00600          DECF    MIN_SEC, F       ;REDUCE SECONDS
0222 0C09           00601          MOVLW   STMR             ;LOAD FSR WITH S_TMR
0223 0024           00602          MOVWF   FSR              ;       /
0224 0955           00603          CALL    INC_60           ;INC SECONDS
0225 0D00           00604          IORLW   0                ;DO AN OPERATION
0226 0743           00605          BTFSS   STATUS,Z         ;IF RETURN = 0 SKIP
0227 0A38           00606          GOTO    CHK_AL_TIM       ;CHK ALRM
                    00607 ;INC MINUTES COUNT
0228 03B6           00608          SWAPF   MIN_SEC, F       ;SWAP MIN SEC
0229 0216           00609          MOVF    MIN_SEC,W        ;GET MIN_SEC IN W
022A 0E0F           00610          ANDLW   B'00001111'      ;MASK SECONDS
022B 0743           00611          BTFSS   STATUS,Z         ;SKIP IF NOT SET
022C 00F6           00612          DECF    MIN_SEC, F       ;ELSE DEC
022D 03B6           00613          SWAPF   MIN_SEC, F       ;SWAP BACK
022E 0966           00614          CALL    CHK_SILNC_TIM    ;SILENCE ON?
022F 0C0A           00615          MOVLW   MTMR             ;INC MINUTES
0230 0024           00616          MOVWF   FSR              ;       /
0231 0955           00617          CALL    INC_60           ;       /
0232 0D00           00618          IORLW   0                ;DO AN OPERATION
0233 0743           00619          BTFSS   STATUS,Z         ;IF 0 THEN SKIP
0234 0A38           00620          GOTO    CHK_AL_TIM       ;CHECK ALARM TIME
                    00621 ;INC HOUR COUNT
0235 0C0B           00622          MOVLW   HTMR             ;GET HTMR IN FSR
0236 0024           00623          MOVWF   FSR
0237 0989           00624          CALL    INC_HR           ;INC HOURS
                    00625 ;
0238                00626 CHK_AL_TIM
0238 0718           00627          BTFSS   ALFLAG,ALONOF    ;IF OFF QUIT
0239 0800           00628          RETLW   0                ;       /
023A 0658           00629          BTFSC   ALFLAG,SILNC     ;RET IF IN SILENCE
023B 0800           00630          RETLW   0
023C 0638           00631          BTFSC   ALFLAG,INAL      ;ALREADY DONE
023D 0A4D           00632          GOTO    CHK_1_MIN        ;SEE IF 1 MIN UP
                    00633 ;         RETLW   0                ;YES THEN QUIT
023E 020D           00634          MOVF    HALARM,W         ;CHK HRS
023F 018B           00635          XORWF   HTMR,W           ;EQUAL?
0240 0743           00636          BTFSS   STATUS,Z         ;YES THEN SKIP
0241 0800           00637          RETLW   0                ;ELSE RET
0242 020C           00638          MOVF    MALARM,W         ;CHK MIN
0243 018A           00639          XORWF   MTMR,W           ;EQUAL?
0244 0743           00640          BTFSS   STATUS,Z         ;YES THEN SKIP
0245 0800           00641          RETLW   0                ;ELSE RET
0246 0209           00642          MOVF    STMR,W           ;SEE IF SEC=0
0247 0743           00643          BTFSS   STATUS,Z         ;YES THEN SKIP
0248 0800           00644          RETLW   0                ;NO THEN RET
0249 0538           00645          BSF     ALFLAG,INAL      ;SET IN ALARM FLAG
024A 0C10           00646          MOVLW   10               ;SET 1 MIN TIMER
024B 0036           00647          MOVWF   MIN_SEC          ;       /
024C 0800           00648          RETLW   0
```


© 1997 Microchip Technology Inc.

```
                  00649 ;
024D              00650 CHK_1_MIN
024D 0396         00651         SWAPF   MIN_SEC,W       ;SWAP IN W
024E 0E0F         00652         ANDLW   B'00001111'     ;CHK MINUTES
024F 0743         00653         BTFSS   STATUS,Z        ;0 THEN SKIP
0250 0800         00654         RETLW   0               ;ELSE RET
0251 0438         00655         BCF     ALFLAG,INAL     ;CLR IN ALARM
0252 0478         00656         BCF     ALFLAG,INAA     ;CLR IN AA
0253 0505         00657         BSF     PORT_A,BEP      ;STOP BEEPER
0254 0800         00658         RETLW   0
                  00659 ;
0255              00660 INC_60
0255 02A0         00661         INCF    F0, F           ;INC AND GET IN W
0256 0200         00662         MOVF    F0,0            ;       /
0257 0E0F         00663         ANDLW   B'00001111'     ;MASK HI BITS
0258 0F0A         00664         XORLW   B'00001010'     ;= 10 THEN MAKE IT 0
0259 0743         00665         BTFSS   STATUS,Z        ;       /
025A 0801         00666         RETLW   1               ;ELSE RETURN NON ZERO
025B 0CF0         00667         MOVLW   B'11110000'     ;ZERO LSB
025C 0160         00668         ANDWF   F0, F           ;  /
025D 03A0         00669         SWAPF   F0, F           ;SWAP INDIRECT
025E 02A0         00670         INCF    F0, F           ;INC
025F 0200         00671         MOVF    F0,0            ;GET IN W
0260 03A0         00672         SWAPF   F0, F           ;SWAP F0 BACK
0261 0F06         00673         XORLW   D'6'            ;=6 THEN SKIP
0262 0743         00674         BTFSS   STATUS,Z        ;       /
0263 0801         00675         RETLW   1               ;ELSE RETURN NZ
0264 0060         00676         CLRF    F0              ;       /
0265 0800         00677         RETLW   0               ;RET 0
                  00678 ;
                  00679     PAGE
                  00680 ;
0266              00681 CHK_SILNC_TIM
0266 0758         00682         BTFSS   ALFLAG,SILNC    ;CHK IF IN SILENCE
0267 0800         00683         RETLW   0               ;NO THEN SKIP
0268 0396         00684         SWAPF   MIN_SEC,W       ;GET MIN IN W
0269 0E0F         00685         ANDLW   B'00001111'     ;MASK SECS
026A 0743         00686         BTFSS   STATUS,Z        ;ZERO?
026B 0800         00687         RETLW   0               ;NO THEN RET
026C 0458         00688         BCF     ALFLAG,SILNC    ;RESET SILENCE
026D 0C10         00689         MOVLW   10              ;SET 1 MIN TIMER
026E 0036         00690         MOVWF   MIN_SEC         ;       /
026F 0800         00691         RETLW   0
                  00692 ;
                  00693 ;
0270              00694 CHK_DE_BOUNCE
0270 06B7         00695         BTFSC   ENTFLG,INKEYBEP ;IN KEY BEEP?
0271 0A76         00696         GOTO    CHK_DEB_1       ;YES THEN DEC TIMER
0272 07B0         00697         BTFSS   FLAG,KEY_BEEP   ;KEY BEEP SET?
0273 0A7F         00698         GOTO    CHK_SERV        ;NO, SEE IF SERVICED
0274 0678         00699         BTFSC   ALFLAG,INAA     ;IN AA?
0275 0A86         00700         GOTO    CHK_BEP_ON      ;YES THEN SEE IF ON
0276              00701 CHK_DEB_1
0276 05B7         00702         BSF     ENTFLG,INKEYBEP ;SET FLAG
0277 0215         00703         MOVF    DEBOUNCE,W      ;GET IN W
0278 0643         00704         BTFSC   STATUS,Z        ;NZ THEN SKIP
0279 0C14         00705         MOVLW   D'20'           ;ELSE DB 100 mSEC
027A 0035         00706         MOVWF   DEBOUNCE        ;       /
027B 0405         00707         BCF     PORT_A,BEP      ;TURN ON BEEPER
027C 02F5         00708         DECFSZ  DEBOUNCE, F     ;DEC AND CHK
027D 0A08         00709         GOTO    UP_TMR_1        ;GO BACK
027E 0505         00710         BSF     PORT_A,BEP      ;TURN OFF BEEPER
027F              00711 CHK_SERV
                  00712 ;       CLRF    DEBOUNCE
                  00713 ;       BSF     PORT_A,BEP
027F 07F0         00714         BTFSS   FLAG,SERVICED   ;SERVICED THEN SKIP
```

```
0280 0A08          00715         GOTO    UP_TMR_1          ;GO BACK
0281 04F0          00716         BCF     FLAG,SERVICED     ;ELSE CLEAR FLAGS
0282 04D0          00717         BCF     FLAG,KEY_HIT      ;       /
0283 04B0          00718         BCF     FLAG,KEY_BEEP     ;RESET FLAG
0284 04B7          00719         BCF     ENTFLG,INKEYBEP   ;    /
0285 0A08          00720         GOTO    UP_TMR_1          ;GO BACK
                   00721 ;
0286               00722 CHK_BEP_ON
0286 0705          00723         BTFSS   PORT_A,BEP        ;IF OFF THEN SKIP
0287 0A08          00724         GOTO    UP_TMR_1          ;ELSE WAIT
0288 0A76          00725         GOTO    CHK_DEB_1         ;RETURN
                   00726 ;
                   00727 ;
0289               00728 INC_HR
0289 02A0          00729         INCF    F0, F             ;INC HOUR TIMER
028A 0200          00730         MOVF    F0,W              ;GET HR TMR IN W
028B 0031          00731         MOVWF   TEMP              ;SAVE IN TEMP
028C 0E0F          00732         ANDLW   B'00001111'       ;CHK LO BYTE = 10
028D 0F0A          00733         XORLW   D'10'             ;       /
028E 0743          00734         BTFSS   STATUS,Z          ;YES THEN SKIP
028F 0A93          00735         GOTO    INC_AM_PM         ;ELSE CHK 12
0290 0C10          00736         MOVLW   B'00010000'       ;LOAD 1 IN MSB
0291 0020          00737         MOVWF   F0
0292 0AA3          00738         GOTO    RESTORE_AM_PM     ;RESTORE AM/PM
0293               00739 INC_AM_PM
0293 04E0          00740         BCF     F0,AM_PM          ;CLEAR AM/PM
0294 0200          00741         MOVF    F0,W              ;GET IN W
0295 0F12          00742         XORLW   12H               ;SEE IF 12 HEX
0296 0743          00743         BTFSS   STATUS,Z          ;YES THEN SKIP
0297 0A9D          00744         GOTO    CHK_13            ;ELSE CHK 13
0298 07F1          00745         BTFSS   TEMP,AM_PM        ;IF SET, SKIP
0299 0A9C          00746         GOTO    SET_AM_PM         ;ELSE SET
029A 04E0          00747         BCF     F0,AM_PM          ;CLEAR FLAG
029B 0800          00748         RETLW   0                 ;RETURN
029C               00749 SET_AM_PM
029C 05E0          00750         BSF     F0,AM_PM          ;SET FLAG
029D               00751 CHK_13
029D 0200          00752         MOVF    F0,W              ;GET IN W
029E 0F13          00753         XORLW   13H               ;SEE IF 13
029F 0743          00754         BTFSS   STATUS,Z          ;YES THEN SKIP
02A0 0AA3          00755         GOTO    RESTORE_AM_PM
02A1               00756 SET_1_HR
02A1 0C01          00757         MOVLW   B'00000001'       ;SET TO 1
02A2 0020          00758         MOVWF   F0
02A3               00759 RESTORE_AM_PM
02A3 06F1          00760         BTFSC   TEMP,AM_PM        ;SKIP IF AM
02A4 05E0          00761         BSF     F0,AM_PM          ;ELSE SET TO PM
02A5 0800          00762         RETLW   0
                   00763 ;
                   00764 ;
                   00765 ;
                   00766     PAGE
0400               00767         ORG     400
                   00768 ;
                   00769 ;KEY DEFINITIONS
  0000000A         00770 ALARM_KEY       EQU     0A
  0000000B         00771 CE_KEY          EQU     0B
  0000000C         00772 SNOOZE_KEY      EQU     0C
  0000000D         00773 AM_PM_KEY       EQU     0D
  0000000E         00774 CLR_ALARM_KEY   EQU     0E
  0000000F         00775 SET_KEY         EQU     0F
                   00776 ;
0400               00777 SERVICE_KEYS
0400 07D0          00778         BTFSS   FLAG,KEY_HIT      ;NO KEY HIT THEN ...
0401 0800          00779         RETLW   0                 ;RETURN
0402 06F0          00780         BTFSC   FLAG,SERVICED     ;IF NOT SERVICED SKIP
```


© 1997 Microchip Technology Inc.

```
0403 0800          00781          RETLW   0               ;ELSE RETURN
0404 05F0          00782          BSF     FLAG,SERVICED   ;SET SERVICED FLAG
0405 0210          00783          MOVF    FLAG,W          ;GET MODE OF OPERATION
0406 0E03          00784          ANDLW   B'00000011'     ;       /
0407 0643          00785          BTFSC   STATUS,Z        ;00 THEN RTM
0408 0A10          00786          GOTO    RTMKS           ;RTM KEY SERVICE
0409 0031          00787          MOVWF   TEMP            ;SAVE IN TEMP
040A 02F1          00788          DECFSZ  TEMP, F         ;REDUCE TEMP
040B 0A0D          00789          GOTO    SK1             ;SKIP
040C 0A1D          00790          GOTO    ATMKS           ;01, DO ALARM MODE
040D               00791 SK1
040D 02F1          00792          DECFSZ  TEMP, F         ;REDUCE TEMP
040E 0800          00793          RETLW   0               ;11 THEN RETURN
040F 0A2A          00794          GOTO    DEMKS           ;10, DATA ENTRY MODE
                   00795 ;
                   00796 ;REAL TIME MODE KEY SERVICE
0410               00797 RTMKS
0410 09BA          00798          CALL    CHK_AL_KEYS     ;CHK ALARM KEYS
0411 0D00          00799          IORLW   0               ;SEE IF NZ RET
0412 0643          00800          BTFSC   STATUS,Z        ;NZ THEN SKIP
0413 0800          00801          RETLW   0               ;ELSE RETURN
0414 0C0F          00802          MOVLW   SET_KEY         ;SEE IF SET KEY
0415 0193          00803          XORWF   NEW_KEY,W       ;       /
0416 0643          00804          BTFSC   STATUS,Z        ;NO THEN SKIP
0417 0A91          00805          GOTO    SERV_SET_RTM    ;SERVICE SET KEY
0418 0C0A          00806          MOVLW   ALARM_KEY       ;ALARM KEY?
0419 0193          00807          XORWF   NEW_KEY,W       ;     /
041A 0643          00808          BTFSC   STATUS,Z        ;NO THEN SKIP
041B 0AAB          00809          GOTO    SERV_ALARM_RTM  ;ELSE SERVICE ALARM
041C               00810 IGNORE_KEY
041C 0800          00811          RETLW   0               ;ELSE RETURN
                   00812 ;
                   00813 ;ALARM TIME MODE KEY SERVICE
041D               00814 ATMKS
041D 09BA          00815          CALL    CHK_AL_KEYS     ;CHECK ALARM KEYS
041E 0D00          00816          IORLW   0               ;CHECK IF 0
041F 0643          00817          BTFSC   STATUS,Z        ;NZ THEN SKIP
0420 0800          00818          RETLW   0               ;ELSE RETURN
0421 0C0F          00819          MOVLW   SET_KEY         ;SEE IF SET KEY
0422 0193          00820          XORWF   NEW_KEY,W       ;       /
0423 0643          00821          BTFSC   STATUS,Z        ;NO THEN SKIP
0424 0A9C          00822          GOTO    SERV_SET_ATM
0425 0C0A          00823          MOVLW   ALARM_KEY       ;GET ALARM KEY
0426 0193          00824          XORWF   NEW_KEY,W       ;SEE IF HIT
0427 0643          00825          BTFSC   STATUS,Z        ;NO THEN SKIP
0428 0AA2          00826          GOTO    SERV_ALARM_ATM  ;ELSE SERVICE
0429 0A1C          00827          GOTO    IGNORE_KEY
                   00828 ;
                   00829 ;DATA ENTRY MODE KEY SERVICE
042A               00830 DEMKS
042A 09BA          00831          CALL    CHK_AL_KEYS     ;CHECK ALARM KEYS
042B 0D00          00832          IORLW   0               ;CHK IF 0
042C 0643          00833          BTFSC   STATUS,Z        ;NZ THEN SKIP
042D 0800          00834          RETLW   0               ;ELSE RETURN
042E 0C0F          00835          MOVLW   SET_KEY         ;IF SET KEY THEN END
042F 0193          00836          XORWF   NEW_KEY,W       ;       /
0430 0643          00837          BTFSC   STATUS,Z        ;NO THEN SKIP
0431 0A3F          00838          GOTO    DEMKS_END       ;GOTO END
0432 0C0B          00839          MOVLW   CE_KEY          ;IF CLEAR ENTRY
0433 0193          00840          XORWF   NEW_KEY,W       ;       /
0434 0643          00841          BTFSC   STATUS,Z        ;SKIP IF NO
0435 0A48          00842          GOTO    DEMKS_END_1     ;ABANDON ENTRY
0436 0737          00843          BTFSS   ENTFLG,HR10     ;10'S HRS DONE?
0437 0A54          00844          GOTO    ENT_HR_10       ;NO THEN GET
0438 0757          00845          BTFSS   ENTFLG,HR       ;HRS DONE?
0439 0A5F          00846          GOTO    ENT_HRS         ;NO THEN GET
```


© 1997 Microchip Technology Inc.

```
043A 0777          00847          BTFSS   ENTFLG,MIN10    ;10'S MIN. DONE?
043B 0A72          00848          GOTO    ENT_MIN_10      ;NO THEN GET
043C 0797          00849          BTFSS   ENTFLG,MIN      ;MIN DONE?
043D 0A7F          00850          GOTO    ENT_MIN         ;NO THEN GET
043E 0A87          00851          GOTO    ENT_AM_PM       ;NO THEN GET
043F               00852 DEMKS_END
043F 0717          00853          BTFSS   ENTFLG,RTATS    ;GET OLD STATUS
0440 0A4D          00854          GOTO    LD_RTM          ;LOAD IN TIME
0441 020E          00855          MOVF    MENTRY,W        ;LD IN ALARM
0442 002C          00856          MOVWF   MALARM          ;       /
0443 020F          00857          MOVF    HENTRY,W        ;     /
0444 002D          00858          MOVWF   HALARM          ;   /
0445 0450          00859          BCF     FLAG,ALRMLED    ;CLEAR FLAG
0446 0618          00860          BTFSC   ALFLAG,ALONOF   ;SEE IF ON-OFF
0447 0550          00861          BSF     FLAG,ALRMLED    ;ELSE SET
0448               00862 DEMKS_END_1
0448 0410          00863          BCF     FLAG,0          ;RTM MODE
0449 0430          00864          BCF     FLAG,1          ;    /
044A 0490          00865          BCF     FLAG,FLASH      ;STOP FLASH
044B               00866 SERV_COM_RET
044B 05B0          00867          BSF     FLAG,KEY_BEEP
044C 0800          00868          RETLW   0               ;RETURN
                   00869 ;
044D               00870 LD_RTM
044D 020E          00871          MOVF    MENTRY,W        ;LD IN RTM
044E 002A          00872          MOVWF   MTMR            ;       /
044F 020F          00873          MOVF    HENTRY,W        ;     /
0450 002B          00874          MOVWF   HTMR            ;   /
0451 0068          00875          CLRF    MSTMR           ;CLR TIME
0452 0069          00876          CLRF    STMR            ;  /
0453 0A48          00877          GOTO    DEMKS_END_1     ;GO BACK
                   00878 ;
0454               00879 ENT_HR_10
0454 0213          00880          MOVF    NEW_KEY,W       ;SEE IF 0
0455 0643          00881          BTFSC   STATUS,Z        ;NZ THEN SKIP
0456 0A5C          00882          GOTO    LD_HENTRY_0     ;LOAD 0
0457 02D3          00883          DECFSZ  NEW_KEY,0       ;1 THE SKIP
0458 0A1C          00884          GOTO    IGNORE_KEY      ;ELSE IGNORE KEY
0459 058F          00885          BSF     HENTRY,4        ;SET TO 1
045A 0537          00886          BSF     ENTFLG,HR10     ;SET FLAG
045B 0A4B          00887          GOTO    SERV_COM_RET    ;GO GET NEXT
045C               00888 LD_HENTRY_0
045C 048F          00889          BCF     HENTRY,4        ;SET TO 0
045D 0537          00890          BSF     ENTFLG,HR10
045E 0A4B          00891          GOTO    SERV_COM_RET    ;
045F               00892 ENT_HRS
045F 0C0F          00893          MOVLW   HENTRY          ;USE INDIRECT ADDR.
0460 0024          00894          MOVWF   FSR             ;       /
0461 068F          00895          BTFSC   HENTRY,4        ;SEE IF 0
0462 0A6D          00896          GOTO    ALLOW0_2        ;YES THEN 0,1&2
0463 0C0A          00897          MOVLW   D'10'           ;SEE IF 0 - 9
0464 0093          00898          SUBWF   NEW_KEY,W       ;       /
0465 0603          00899          BTFSC   STATUS,C        ;IF C THEN SKIP
0466 0A1C          00900          GOTO    IGNORE_KEY      ;ELSE IGNORE
0467               00901 ENT_LO_COM1
0467 0557          00902          BSF     ENTFLG,HR       ;SET FLAG
0468               00903 ENT_LO_COM
0468 0200          00904          MOVF    F0,W            ;LD HRS
0469 0EF0          00905          ANDLW   B'11110000'     ;MASK LO NIBL
046A 0113          00906          IORWF   NEW_KEY,W       ;OR NEW KEY
046B 0020          00907          MOVWF   F0              ;SAVE BACK
046C 0A4B          00908          GOTO    SERV_COM_RET    ;GET NEXT
046D               00909 ALLOW0_2
046D 0C03          00910          MOVLW   D'3'            ;SEE IF 0 - 2
046E 0093          00911          SUBWF   NEW_KEY,W       ;       /
046F 0603          00912          BTFSC   STATUS,C        ;<3 THEN SKIP
```


© 1997 Microchip Technology Inc.

```
0470 0A1C          00913          GOTO    IGNORE_KEY
0471 0A67          00914          GOTO    ENT_LO_COM1     ;       /
                   00915 ;
0472               00916 ENT_MIN_10
0472 0C0E          00917          MOVLW   MENTRY          ;DO INDIRECT ADDR.
0473 0024          00918          MOVWF   FSR             ;       /
0474 0C06          00919          MOVLW   D'6'            ;ALLOW 0 - 5
0475 0093          00920          SUBWF   NEW_KEY,W       ;       /
0476 0603          00921          BTFSC   STATUS,C        ;IF C THEN SKIP
0477 0A1C          00922          GOTO    IGNORE_KEY      ;ELSE IGNORE
0478 0380          00923          SWAPF   F0,W            ;SWAP AND GET
0479 0EF0          00924          ANDLW   B'11110000'     ;MASK LO NIBL
047A 0113          00925          IORWF   NEW_KEY,W       ;OR NEW KEY
047B 0020          00926          MOVWF   F0              ;SAVE BACK
047C 03A0          00927          SWAPF   F0, F           ;SWAP BACK
047D 0577          00928          BSF     ENTFLG,MIN10
047E 0A4B          00929          GOTO    SERV_COM_RET    ;GET NEXT
                   00930 ;
047F               00931 ENT_MIN
047F 0C0E          00932          MOVLW   MENTRY          ;DO INDIRECT
0480 0024          00933          MOVWF   FSR             ;       /
0481 0C0A          00934          MOVLW   D'10'           ;ALLOW 0 - 9
0482 0093          00935          SUBWF   NEW_KEY,W       ;SEE IF >
0483 0603          00936          BTFSC   STATUS,C        ;NO THEN SKIP
0484 0A1C          00937          GOTO    IGNORE_KEY      ;ELSE IGNORE
0485 0597          00938          BSF     ENTFLG,MIN      ;SET FLAG
0486 0A68          00939          GOTO    ENT_LO_COM      ;       /
                   00940 ;
0487               00941 ENT_AM_PM
0487 0C0D          00942          MOVLW   AM_PM_KEY       ;AM/PM KEY?
0488 0193          00943          XORWF   NEW_KEY,W       ;     /
0489 0743          00944          BTFSS   STATUS,Z        ;YES THEN SKIP
048A 0A1C          00945          GOTO    IGNORE_KEY
048B 07EF          00946          BTFSS   HENTRY,AM_PM    ;TEST BIT
048C 0A8F          00947          GOTO    SETAMPM         ;ELSE SET
048D 04EF          00948          BCF     HENTRY,AM_PM    ;CLEAR FLAG
048E 0A4B          00949          GOTO    SERV_COM_RET    ;GOTO END
048F               00950 SETAMPM
048F 05EF          00951          BSF     HENTRY,AM_PM    ;SET FLAG
0490 0A4B          00952          GOTO    SERV_COM_RET
                   00953 ;
                   00954      PAGE
                   00955 ;
0491               00956 SERV_SET_RTM
0491 020A          00957          MOVF    MTMR,W          ;TRANSFER TIME
0492 002E          00958          MOVWF   MENTRY          ;TO DATA ENTRY
0493 020B          00959          MOVF    HTMR,W          ;       /
0494 002F          00960          MOVWF   HENTRY          ;     /
0495               00961 SERV_COM
0495 0210          00962          MOVF    FLAG,W          ;SAVE IN W
0496 0E01          00963          ANDLW   B'00000001'     ;ATM OR RTM MODE?
0497 0037          00964          MOVWF   ENTFLG          ;SAVE IN ENTFLG
0498 0CF2          00965          MOVLW   B'11110010'     ;FORCE 1S
0499 0130          00966          IORWF   FLAG, F         ;     /
049A 0410          00967          BCF     FLAG,0          ;   /
049B 0800          00968          RETLW   0
                   00969 ;
049C               00970 SERV_SET_ATM
049C 020C          00971          MOVF    MALARM,W        ;TRANSFER ALARM
049D 002E          00972          MOVWF   MENTRY          ;TO DATA ENTRY
049E 020D          00973          MOVF    HALARM,W        ;       /
049F 002F          00974          MOVWF   HENTRY          ;     /
04A0 0518          00975          BSF     ALFLAG,ALONOF   ;SET FLAG
04A1 0A95          00976          GOTO    SERV_COM        ;GOTO COMMON
                   00977 ;
04A2               00978 SERV_ALARM_ATM
```


© 1997 Microchip Technology Inc.

```
04A2 0718          00979          BTFSS   ALFLAG,ALONOF   ;TEST ON/OFF
04A3 0AA6          00980          GOTO    SET_ALONOF      ;SET ON/OF FLG
04A4 0418          00981          BCF     ALFLAG,ALONOF   ;CLEAR FLAG
04A5 0AA7          00982          GOTO    SERV_ATM_COM    ;RET THRO COM
04A6               00983 SET_ALONOF
04A6 0518          00984          BSF     ALFLAG,ALONOF   ;SET FLAG
04A7               00985 SERV_ATM_COM
04A7 05B0          00986          BSF     FLAG,KEY_BEEP   ;BEEP
04A8 0CF0          00987          MOVLW   B'11110000'     ;CLEAR SEC COUNT
04A9 0176          00988          ANDWF   MIN_SEC, F      ;       /
04AA 0800          00989          RETLW   0               ;RETURN
                   00990 ;
04AB               00991 SERV_ALARM_RTM
04AB 05B0          00992          BSF     FLAG,KEY_BEEP   ;SET BEEP FLAG
04AC 0510          00993          BSF     FLAG,0          ;SET TO ALARM TIME
04AD 0430          00994          BCF     FLAG,1          ;       /
04AE 0C05          00995          MOVLW   D'05'           ;SAVE 5 IN MIN_SEC
04AF 0036          00996          MOVWF   MIN_SEC         ;       /
04B0 0800          00997          RETLW   0
                   00998 ;
04B1               00999 SERV_SNOOZE
04B1 0CA0          01000          MOVLW   0A0             ;SNOOZE FOR 10 MINS
04B2 0036          01001          MOVWF   MIN_SEC         ;       /
04B3 0558          01002          BSF     ALFLAG,SILNC    ;SET FLAG
04B4               01003 CLR_AL_COM
04B4 05B0          01004          BSF     FLAG,KEY_BEEP   ;SET BEEP FLAG
04B5 007A          01005          CLRF    AATMR           ;RESET AA TIMER
04B6 0079          01006          CLRF    AAFLAG          ;CLEAR AA FLAGS
04B7 0478          01007          BCF     ALFLAG,INAA     ;RESET INAA FLAG
04B8 0505          01008          BSF     PORT_A,BEP      ;TURN OFF BEEPER
04B9 0800          01009          RETLW   0               ;RET
                   01010 ;
04BA               01011 CHK_AL_KEYS
04BA 0718          01012          BTFSS   ALFLAG,ALONOF   ;ALARM ON?
04BB 0801          01013          RETLW   1               ;NO THEN RET
04BC 0738          01014          BTFSS   ALFLAG,INAL     ;IN ALARM?
04BD 0801          01015          RETLW   1               ;NO THEN SKIP
04BE 0C0E          01016          MOVLW   CLR_ALARM_KEY   ;CHECK IF CLR ALARM
04BF 0193          01017          XORWF   NEW_KEY,W       ;       /
04C0 0643          01018          BTFSC   STATUS,Z        ;NO THEN SKIP
04C1 0AC7          01019          GOTO    CLR_ALARM       ;ELSE CLEAR ALARM
04C2 0C0C          01020          MOVLW   SNOOZE_KEY      ;SEE IF SNOOZE HIT
04C3 0193          01021          XORWF   NEW_KEY,W       ;       /
04C4 0743          01022          BTFSS   STATUS,Z        ;YES THEN SKIP
04C5 0801          01023          RETLW   1
04C6 0AB1          01024          GOTO    SERV_SNOOZE
                   01025 ;
04C7               01026 CLR_ALARM
04C7 0438          01027          BCF     ALFLAG,INAL     ;CLEAR ALARM
04C8 0458          01028          BCF     ALFLAG,SILNC    ;CLEAR SILENCE
04C9 0C0F          01029          MOVLW   B'00001111'     ;CLEAR MINS
04CA 0176          01030          ANDWF   MIN_SEC, F      ;       /
04CB 0AB4          01031          GOTO    CLR_AL_COM
                   01032 ;
0600               01033          ORG     600
                   01034 ;If the AA alarm is set, then this routine takes care of
                   01035 ;the timing in sounding the alarm.
                   01036 ;
0600               01037 SOUND_AA
0600 0738          01038          BTFSS   ALFLAG,INAL     ;SKIP IF IN ALRM
0601 0800          01039          RETLW   0               ;ELSE RETURN
0602 0658          01040          BTFSC   ALFLAG,SILNC    ;SKIP IF NOT IN SILENCE
0603 0800          01041          RETLW   0               ;ELSE RET
0604 06B7          01042          BTFSC   ENTFLG,INKEYBEP ;SKIP IF NOT IN KEY BEEP
0605 0A55          01043          GOTO    CHK_COLSN       ;CHK COLLISION
0606               01044 SND_AA_0
```


© 1997 Microchip Technology Inc.

```
0606 0778          01045          BTFSS   ALFLAG,INAA     ;SKIP IF IN AA
0607               01046 SND_AA_1
0607 0919          01047          CALL    INIT_AA         ;INIT ALL
0608 0719          01048          BTFSS   AAFLAG,0        ;SKIP IF DONE
0609 0A21          01049          GOTO    DO_CYCL0        ;DO FIRST CYCL
060A 0739          01050          BTFSS   AAFLAG,1        ;SKIP IF DONE
060B 0A29          01051          GOTO    DO_CYCL1        ;ELSE 2ND CYCLE
060C 0759          01052          BTFSS   AAFLAG,2        ;SKIP IF DONE
060D 0A31          01053          GOTO    DO_CYCL2        ;ELSE DO 3RD CYCLE
060E 0779          01054          BTFSS   AAFLAG,3        ;SKIP IF DONE
060F 0A39          01055          GOTO    DO_CYCL3        ;DO CYCLE 4
0610 0799          01056          BTFSS   AAFLAG,4        ;SKIP IF DONE
0611 0A3E          01057          GOTO    DO_CYCL4        ;DO CYCLE 5
0612 07B9          01058          BTFSS   AAFLAG,5        ;SKIP IF DONE
0613 0A43          01059          GOTO    DO_CYCL5        ;DO CYCLE 6
0614 07D9          01060          BTFSS   AAFLAG,6        ;SKIP IF DONE
0615 0A48          01061          GOTO    DO_CYCL6        ;DO CYCLE 6
0616 07F9          01062          BTFSS   AAFLAG,7        ;SKIP IF DONE
0617 0A50          01063          GOTO    DO_CYCL7        ;DO CYCLE 7
0618 0A07          01064          GOTO    SND_AA_1        ;GO BACK
                   01065 ;
0619               01066 INIT_AA
0619 0079          01067          CLRF    AAFLAG          ;CLEAR ALL FLAGS
061A 0578          01068          BSF     ALFLAG,INAA     ;SET IN AA FLAG
061B 0A2D          01069          GOTO    PUT_ON_100      ;ON 100 MSECS
                   01070 ;
061C               01071 DEC_AA_TMR
061C 00FA          01072          DECF    AATMR, F        ;REDUCE TIMER
061D 021A          01073          MOVF    AATMR,W         ;GET IN W
061E 0743          01074          BTFSS   STATUS,Z        ;CHECK IF Z
061F 0801          01075          RETLW   1               ;NO THEN NZ
0620 0800          01076          RETLW   0               ;ELSE 0
                   01077 ;
0621               01078 DO_CYCL0
0621 091C          01079          CALL    DEC_AA_TMR      ;REDUCE TIMER
0622 0743          01080          BTFSS   STATUS,Z        ;IF NZ THEN RET
0623 0800          01081          RETLW   0
0624 0519          01082          BSF     AAFLAG,0        ;SET DONE FLAG
0625               01083 PUT_OFF_100
0625 0505          01084          BSF     PORT_A,BEP      ;TURN OFF BEEPER
0626 0C14          01085          MOVLW   D'20'           ;FOR 100 MSECS
0627 003A          01086          MOVWF   AATMR           ;       /
0628 0800          01087          RETLW   0
                   01088 ;
0629               01089 DO_CYCL1
0629 091C          01090          CALL    DEC_AA_TMR      ;REDUCE TIMER
062A 0743          01091          BTFSS   STATUS,Z        ;IF NZ THEN RET
062B 0800          01092          RETLW   0
062C 0539          01093          BSF     AAFLAG,1        ;SET DONE FLAG
062D               01094 PUT_ON_100
062D 0405          01095          BCF     PORT_A,BEP      ;TURN ON BEEPER
062E 0C14          01096          MOVLW   D'20'           ;FOR 100 MSECS
062F 003A          01097          MOVWF   AATMR           ;       /
0630 0800          01098          RETLW   0
                   01099 ;
0631               01100 DO_CYCL2
0631 091C          01101          CALL    DEC_AA_TMR      ;REDUCE TIMER
0632 0743          01102          BTFSS   STATUS,Z        ;IF NZ THEN RET
0633 0800          01103          RETLW   0               ;       /
0634 0559          01104          BSF     AAFLAG,2        ;SET DONE FLAG
0635 0505          01105          BSF     PORT_A,BEP      ;TURN OFF BEEPER
0636 0C64          01106          MOVLW   D'100'          ;FOR 500 MSECS
0637 003A          01107          MOVWF   AATMR           ;       /
0638 0800          01108          RETLW   0
                   01109 ;
0639               01110 DO_CYCL3
```

```
0639 091C          01111          CALL    DEC_AA_TMR      ;REDUCE TIMER
063A 0743          01112          BTFSS   STATUS,Z        ;IF NZ THEN RET
063B 0800          01113          RETLW   0               ;       /
063C 0579          01114          BSF     AAFLAG,3        ;SET DONE FLAG
063D 0A2D          01115          GOTO    PUT_ON_100      ;DO NEXT CYCLE
                   01116 ;
063E               01117 DO_CYCL4
063E 091C          01118          CALL    DEC_AA_TMR      ;REDUCE TIMER
063F 0743          01119          BTFSS   STATUS,Z        ;IF NZ THEN RET
0640 0800          01120          RETLW   0               ;       /
0641 0599          01121          BSF     AAFLAG,4        ;SET DONE FLAG
0642 0A25          01122          GOTO    PUT_OFF_100     ;DO NEXT CYCLE
                   01123 ;
0643               01124 DO_CYCL5
0643 091C          01125          CALL    DEC_AA_TMR      ;REDUCE TIMER
0644 0743          01126          BTFSS   STATUS,Z        ;IF NZ THEN RET
0645 0800          01127          RETLW   0               ;       /
0646 05B9          01128          BSF     AAFLAG,5        ;SET DONE FLAG
0647 0A2D          01129          GOTO    PUT_ON_100      ;DO NEXT CYCLE
                   01130 ;
0648               01131 DO_CYCL6
0648 091C          01132          CALL    DEC_AA_TMR      ;REDUCE TIMER
0649 0743          01133          BTFSS   STATUS,Z        ;IF NZ THEN RET
064A 0800          01134          RETLW   0               ;       /
064B 05D9          01135          BSF     AAFLAG,6        ;SET DONE FLAG
064C 0505          01136          BSF     PORT_A,BEP      ;TURN OFF BEEPER
064D 0CC8          01137          MOVLW   D'200'          ;FOR 1000 MSECS
064E 003A          01138          MOVWF   AATMR           ;       /
064F 0800          01139          RETLW   0
                   01140 ;
0650               01141 DO_CYCL7
0650 091C          01142          CALL    DEC_AA_TMR      ;REDUCE TIMER
0651 0743          01143          BTFSS   STATUS,Z        ;IF NZ THEN RET
0652 0800          01144          RETLW   0               ;       /
0653 05F9          01145          BSF     AAFLAG,7        ;SET DONE FLAG
0654 0A2D          01146          GOTO    PUT_ON_100      ;DO NEXT CYCLE
                   01147 ;
0655               01148 CHK_COLSN
0655 0605          01149          BTFSC   PORT_A,BEP      ;IF ON THEN SKIP
0656 0A06          01150          GOTO    SND_AA_0        ;ELSE RET
0657 021A          01151          MOVF    AATMR,W         ;GET TIMER
0658 0643          01152          BTFSC   STATUS,Z        ;NZ THEN SKIP
0659 0A5C          01153          GOTO    LD_AAT_1        ;LOAD A 1 IN TMR
065A 00FA          01154          DECF    AATMR, F        ;REDUCE TIMER
065B 0800          01155          RETLW   0               ;RETURN
065C               01156 LD_AAT_1
065C 02BA          01157          INCF    AATMR, F        ;INC TIMER
065D 0800          01158          RETLW   0               ;RET
                   01159 ;
07FF               01160          ORG     PIC57
07FF               01161 SYS_RESET
07FF 0A00          01162          GOTO    START
                   01163 ;
                   01164          END
```


© 1997 Microchip Technology Inc.

```
0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXX------- ---------------- ----------------
0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0280 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXX---------- ----------------
0400 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0440 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0480 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
04C0 : XXXXXXXXXXXX---- ---------------- ---------------- ----------------
0600 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0640 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXX-- ---------------- ----------------
07C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   746
Program Memory Words Free:  1302


Errors    :     0
Warnings  :     0 reported,     0 suppressed
Messages  :     0 reported,     6 suppressed
```

2

© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Intelligent Remote Positioner (Motor Control)

Author: *Steven Frank*
Vesta Technology Inc.

INTRODUCTION

The excellent cost/performance ratio of the PIC16C5X makes it well suited as a low-cost proportional D.C. actuator controller. This application note depicts a design for a remote intelligent positioning system using a D.C. motor (up to 1/3 hp) run from 12V to 24V. The position accuracy is one in eight bits or 0.4%. The PIC16C5X receives its command and control information via a Microwire® serial bus. However, any serial communication method is applicable.

IMPLEMENTATION

The PIC16C5X based controller receives movement commands from a host, compares them to the actual position, calculates the desired motor drive level and then pulses a full H-bridge (Figure 2). In this way it serves as a remote intelligent positioner, driving the load until it has reached the commanded position. It can be used to control any proportional D.C. actuator (i.e., D.C. motor or proportional valve).

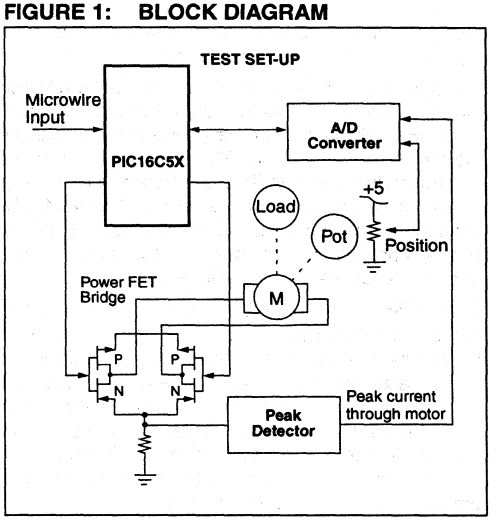
This system is ideally suited for remotely positioned valves and machinery. It can be used with D.C. motors to easily automate manual equipment. Because of the 5-wire serial interface, the positioner can be installed near its power supply and load. The remote intelligent positioner can then be linked to the central control processor by a small diameter easily routed cable. Since the positioner is running its own closed-loop PID algorithm (Figure 3), the host central processor needs only to send position commands and is therefore free to service the user interface, main application software and command multiple remote positioners.

The limit switch inputs provide a safety net which keeps the system from destroying itself in the event that the feedback device is damaged. The optional current sense input can be used to determine if the load has jammed and prevent overheating of the actuator and drive electronics.

The commanded positions are presented to the PIC16C5X via a microwire type protocol at bit-rates of up to 50 Kbs for a 4 MHz part. As currently implemented in this application note, the position request is the only communication. There are several variable locations available which could be used to down-load the loop gain parameters, read positioner information, or set a current limit. The host that is sending the position request must set the chip select low, and wait for the PIC16C5X to raise the "busy" (DO) line high. At this point, eight data bits can be clocked into the PIC16C5X. The requested position is sent most significant bit first and can be any 8-bit value. Values 1 through 255 represent valid positions with 0 being reserved for drive disable.

The PIC16C5X acquires its data by way of a Microwire® A/D converter. This part was chosen for low cost while providing adequate performance. In Figure 1 the second channel of the A/D converter is shown connected to a peak current detector. If the user desires, the PIC16C5X could monitor and protect the motor from overcurrent conditions by monitoring the second channel.

FIGURE 1: BLOCK DIAGRAM



Microwire is a registered trademark of National Semiconductor Corporation.

© 1997 Microchip Technology Inc.

The H-bridge power amplifier will deliver 10 or more amps at upto 24V when properly heat-sinked. It is wired for a modified 4-quadrant mode of operation. One leg of the bridge is used to control direction and the other leg pulses the low FET and the high FET alternately to generate the desired duty-cycle. In this way the system will operate well to produce a desired "speed" without the use of a separate speed control loop. This allows use of the PIC16C5X to control the PID algorithm for position directly while having reasonable speed control. The capacitance at the gates of the FETs combined with the impedance of the drive circuits provides for turn-off of the upper FET before the lower FET turns on... an important criteria.

The PID algorithm itself is where most of the meat of this application note is located so let's look at it more closely. The algorithm is formed by summing the contribution of three basic components. The first calculation is the error upon which the other terms are based.

The **error** is the requested position minus the actual position. It is a signed number whose magnitude can be 255. In order not to lose resolution, the error is stored as an 8-bit magnitude with the sign stored separately in the FLAGS register under ER_SGN. This allows us to resolve a full signed 8-bit error with 8-bit math.

The **proportional term** is merely the algebraic difference of the requested position minus the actual position. It is scaled by a gain term (K_P) called the "proportional gain". The sign (+,-) of this term is important for it tells the system which direction it must drive to correct the error. The proportional term is limited to ±100. Increasing the proportional gain term will improve the dynamic and static accuracy of the system. Increasing it too much will cause oscillations.

The next term that gets calculated is the **integral term (K_I)**. This term is traditionally formed by integrating the error over time. In this application it is done by integrating the K_I term over time. When the error is zero, no integration is performed. This is a more practical way to handle a potentially large number in 8-bit math. By increasing the K_I term the D.C. or static gain of the system is improved. Increasing the integral gain too much can lead to low frequency oscillations.

The **differential term** (K_D) is a stabilizing term that helps keep the integral and proportional terms from overdriving the system through the desired position and thus creating oscillations. As you use more proportional and integral gain you will need more differential gain as well. The differential gain is calculated by looking at the rate of change of the positional error with respect to time. It is actually formed as "delta error/delta time" with the delta time being a program cycle.

The three terms (proportional, integral, differential) are summed algebraically and scaled to produce a percentage speed request between 0 and 100%. The sign of the sum is used to control the direction of the H-bridge. The loop calculations run approximately 20 times per second on a 4 MHz part. This yields sufficient gain-bandwidth for most positioning applications. If higher system performance is desired, the number of pulses can be reduced to 20 and a 16 MHz PIC16C5X can be used. Your loop gains (K_P, K_I, K_D) will have to be recalculated, but the system sample rate will be increased to 400 Hz. This should be sufficient to control a system that has a response time of 20 milliseconds or more.

The key to using the PIC16C5X series parts for PID control and PWM generation is to separate the two into separate tasks. There simply is not the hardware support or the processing speed to accurately do both concurrently. It is fortunate therefore that it is not necessary to do both concurrently. Most systems can be stabilized with a much lower information update rate than the PWM frequency. This supports the approach of calculating the desired percentage, outputting the PWM for a period of time and then recalculating the new desired percentage. Using this technique, the inexpensive PIC16C5X can implement PID control, PWM generation, and will still have processing time left over for monitor or communication functions.

About the Author:

Steven Frank has been designing analog and digital control systems for ten years. His background is in medical and consumer electronics. He has received numerous patents in control systems and instrumentation. At Vesta Technology Inc., Mr. Frank works with a number of engineers on custom embedded control systems designs. Vesta Technology Inc. is a provider of embedded control systems from an array of standard products and designs. Vesta offers custom design services and handles projects from concept to manufacturing.

© 1997 Microchip Technology Inc.

FIGURE 2: PROGRAM FLOWCHART

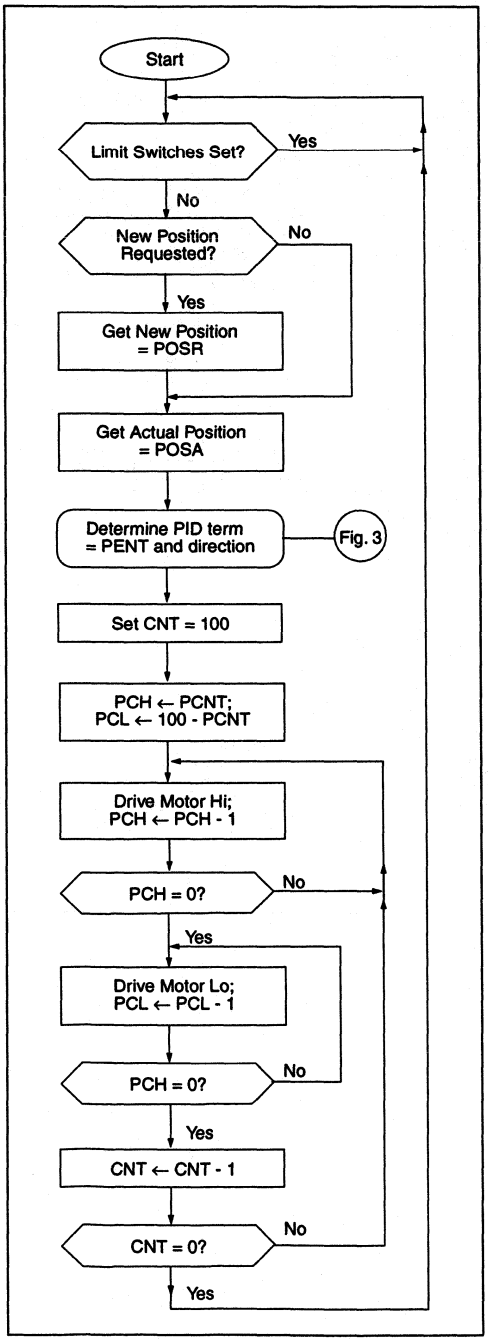
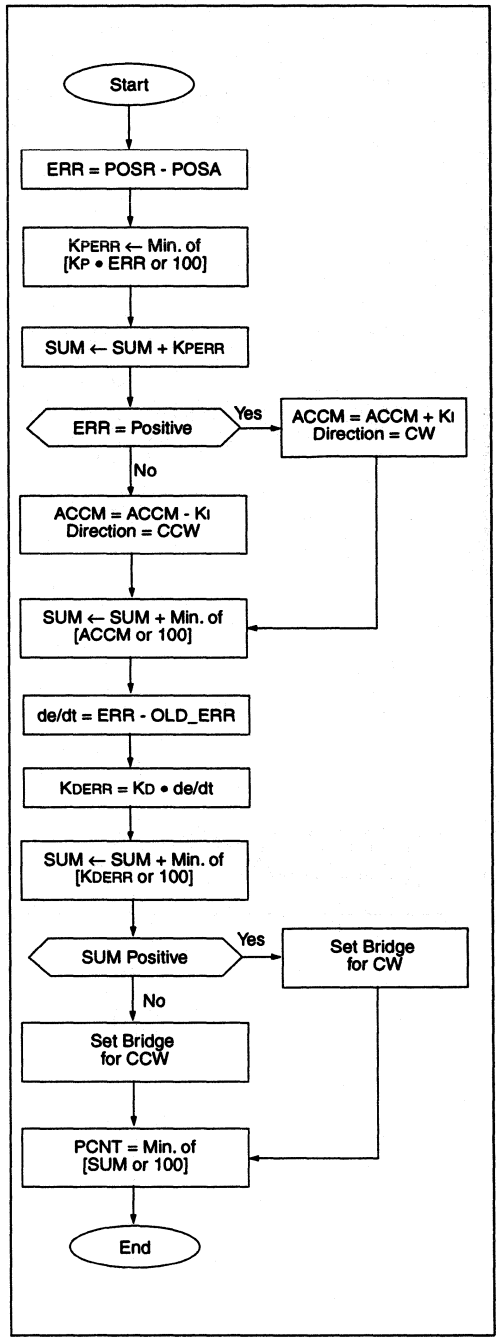
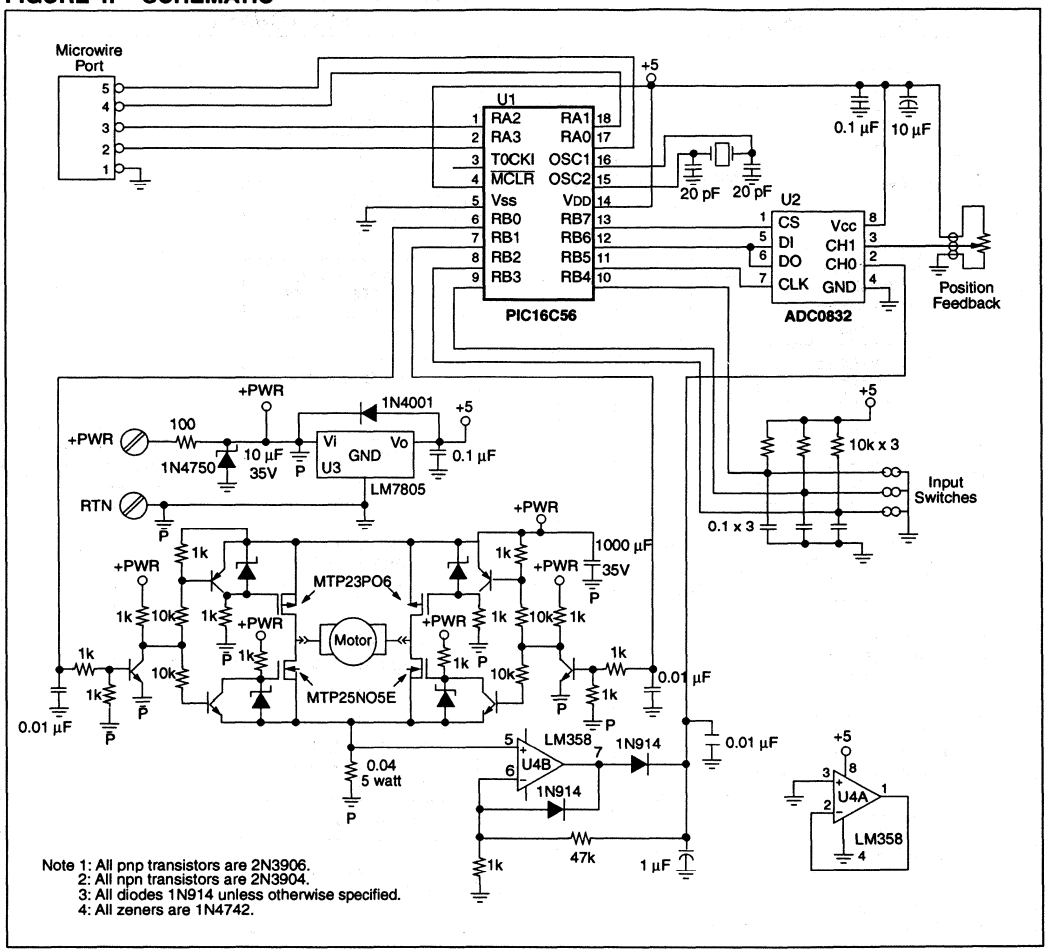


FIGURE 3: PID ALGORITHM FLOWCHART



2

FIGURE 4: SCHEMATIC



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40 Released            MWPOS.ASM   1-16-1997  13:16:02         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001    TITLE " MicroWire Positioner "
                     00002 ;
                     00003 ;        mw8pos.asm
                     00004 ;
                     00005         LIST P=16C56
                     00006 ;
                     00007 ;*****************************************************************
                     00008 ;
                     00009 ;       Program:          MWPOS.ASM
                     00010 ;       Revision Date:    1/10/92 srf   REV. A
                     00011 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00012 ;
                     00013 ;*************************************************************************************
                     00014 ;
                     00015 ;REGISTER EQUATES
                     00016 ;
00000000             00017 PNTR    EQU     00H             ; CONTENTS OF POINTER
00000019             00018 FLAGS   EQU     19H             ; USE THIS VARIABLE LOCATION AS FLAGS
                     00019                                 ; 0 BIT IS SIGN OF ERROR 1 IS NEGATIVE
                     00020                                 ; 1 BIT IS SIGN OF ERROR ACCUMULATOR
                     00021                                 ; 2 BIT IS SIGN OF THE DE/DT TERM
                     00022                                 ; 3 BIT IS DIRECTION 0 IS CW
                     00023                                 ; 4 BIT IS SIGN OF THE OLD ERROR
00000003             00024 STATUS  EQU     03H
00000001             00025 F       EQU     1
00000000             00026 W       EQU     0
00000003             00027 SWR     EQU     03H             ; STATUS WORD REGISTER
                     00028                                 ; 0 = CARRY
                     00029                                 ; 1 = DC
                     00030                                 ; 2 = Z, SET IF RESULT IS ZERO
00000004             00031 FSR     EQU     04H             ; FILE SELECT REGISTER
00000005             00032 PORTA   EQU     05H             ; I/O REG (A0-A3), (A4-A7 DEF=0)
00000006             00033 PORTB   EQU     06H             ; I/O REGISTER(B0-B7)
00000007             00034 HI      EQU     07H             ; NUMBER OF HIGH MICROSECONDS
00000008             00035 LO      EQU     08H             ; NUMBER OF LOW MICROSECONDS
00000009             00036 PCNT    EQU     09H             ; PERCENT DUTYCYCLE REQUEST
0000000A             00037 HI_T    EQU     0AH             ; COUNTER FOR USECONDS LEFT/PULSE HI
0000000B             00038 LO_T    EQU     0BH             ; COUNTER FOR USECONDS LEFT/PULSE LO
0000000C             00039 ERR1    EQU     0CH             ; HOLDER FOR THE POSITIONAL ERROR
                     00040                                 ; THIS IS AN 8 BIT MAGNITUDE WITH THE SIGN
                     00041                                 ; KEPT IN THE FLAG REGISTER (9BIT SIGNED)
0000000D             00042 SUMLO   EQU     0DH             ; PROGRESSIVE SUM OF THE PID TERMS
0000000E             00043 ACCUM   EQU     0EH             ; ERROR ACCUMULATOR
0000000F             00044 ERR_O   EQU     0FH             ; ERROR HISTORY USED FOR de/dt
                     00045                                 ; THIS IS AN 8 BIT MAGNITUDE WITH THE SIGN
                     00046                                 ; KEPT IN THE FLAG REGISTER (9BIT SIGNED)
00000010             00047 POSR    EQU     10H             ; POSITIONAL REQUEST
00000011             00048 POSA    EQU     11H             ; ACTUAL POSITION
00000012             00049 CYCLES  EQU     12H             ; COUNTER FOR CYCLES OUT
                     00050
00000013             00051 mulcnd  equ     13H             ; 8 bit multiplicand
00000013             00052 ACCaLO  EQU     13H             ; same location used for the add routine
00000014             00053 mulplr  equ     14H             ; 8 bit multiplier
00000014             00054 ACCbLO  EQU     14H             ; same location used for the add routine
00000015             00055 H_byte  equ     15H             ; High byte of the 16 bit result
00000015             00056 ACCaHI  EQU     15H             ; same location used for the add routine
00000016             00057 L_byte  equ     16H             ; Low byte of the 16 bit result
00000016             00058 ACCbHI  EQU     16H             ; same location used for the add routine
00000017             00059 count   equ     17H             ; loop counter
00000018             00060 SUMHI   EQU     18H             ; HIGH BYTE OF THE LOOP SUM
                     00061
                     00062
                     00063 ; PORT ASSIGNMENTS AND CONSTANTS
                     00064
00000000             00065 PWMCW   EQU     0               ; CLOCKWISE PWM OUTPUT BIT
00000001             00066 PWMCCW  EQU     1               ; COUNTERCLOCKWISE PWM OUTPUT BIT
00000000             00067 CARRY   EQU     0               ; CARRY BIT IN THE STATUS REGISTER
00000002             00068 Z       EQU     2               ; THE ZERO BIT OF THE STATUS REGISTER
00000001             00069 Same    equ     1               ;
00000000             00070 ER_SGN  EQU     0               ; SIGN BIT FOR THE ERROR IN FLAG REGISTER
```

```
00000001          00071 AC_SGN   EQU     1               ; SIGN BIT FOR THE ERROR ACCUMULATOR
00000002          00072 DE_SGN   EQU     2               ; SIGN BIT FOR DE/DT
00000004          00073 OER_SGN  EQU     4               ; SIGN BIT FOR THE OLD ERROR
00000030          00074 KP       EQU     30              ; PROPORTIONAL GAIN
00000002          00075 KI       EQU     2               ; INTEGRAL GAIN
00000020          00076 KD       EQU     20              ; DIFFERENTIAL GAIN
00000003          00077 DIR      EQU     3               ; THE DIRECTION FLAG
00000007          00078 CSN      EQU     7               ; CHIP SELECT NOT ON A/D
00000006          00079 BV       EQU     6               ; DATA LINE FOR THE A/D
00000005          00080 CK       EQU     5               ; CLOCK LINE FOR THE A/D
00000002          00081 MWDO     EQU     2               ; MICROWIRE DATA OUT FROM POSITIONER
00000001          00082 MWDI     EQU     1               ; MICROWIRE DATA IN TO POSITIONER
00000000          00083 MWCS     EQU     0               ; MICROWIRE CHIP SELECT TO POSITIONER
00000003          00084 MWCK     EQU     3               ; MICROWIRE CLOCK IN TO POSITIONER
                  00085
                  00086
                  00087 ;***** MACROS ********************************************
                  00088 ;
                  00089 CLKUP    MACRO                   ; clock up macro for the microwire
                  00090          BSF     PORTB,CK        ; data acquisition from the a/d
                  00091          NOP
                  00092          ENDM
                  00093
                  00094 CLKDN    MACRO                   ; clock down macro for the microwire
                  00095          BCF     PORTB,CK        ; data acquisition from the a/d
                  00096          NOP
                  00097          ENDM
                  00098
                  00099 GET_BIT  MACRO                   ; ** FOR RECEIVING A/D DATA **
                  00100          BCF     SWR,CARRY
                  00101          BSF     PORTB,CK        ; SET CLOCK BIT HIGH
                  00102          BTFSC   PORTB,BV        ; LOOK AT DATA COMING IN
                  00103          BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
                  00104          RLF     POSA, F         ; ROTATE THE W REG LEFT
                  00105          BCF     PORTB,CK        ; SET THE CLOCK LOW
                  00106          NOP                     ; DELAY
                  00107          ENDM
                  00108
0000 0B88         00109          GOTO    CLRREG
                  00110
                  00111 ;***** MATH ROUTINES ***************************************
                  00112 ;
                  00113
                  00114 ; **** 8 BIT MULTIPLY ********
                  00115 ; ****************************          Begin Multiplier Routine
0001 0075         00116 mpy_S    clrf    H_byte
0002 0076         00117          clrf    L_byte
0003 0C08         00118          movlw   8
0004 0037         00119          movwf   count
0005 0213         00120          movf    mulcnd,W
0006 0403         00121          bcf     STATUS,CARRY    ; Clear the carry bit in the status Reg.
0007 0334         00122 loop     rrf     mulplr, F
0008 0603         00123          btfsc   STATUS,CARRY
0009 01F5         00124          addwf   H_byte,Same
000A 0335         00125          rrf     H_byte,Same
000B 0336         00126          rrf     L_byte,Same
000C 02F7         00127          decfsz  count, F
000D 0A07         00128          goto    loop
000E 0800         00129          retlw   0
                  00130
                  00131 ; ****************************
                  00132 ; DOUBLE PRECISION ADD AND SUBTRACT ( ACCb-ACCa->ACCb )
                  00133
000F 0917         00134 D_sub    call    neg_A           ; At first negate ACCa, then add
                  00135
                  00136 ;***************
                  00137 ; Double Precision Addition ( ACCb+ACCa->ACCb )
                  00138
0010 0213         00139 D_add    movf    ACCaLO,W
0011 01F4         00140          addwf   ACCbLO, F       ; add lsb
0012 0603         00141          btfsc   STATUS,CARRY    ; add in carry
0013 02B6         00142          incf    ACCbHI, F
0014 0215         00143          movf    ACCaHI,W
0015 01F6         00144          addwf   ACCbHI, F       ; add msb
0016 0800         00145          retlw   00
                  00146 ;
                  00147 ;
0017 0273         00148 neg_A    comf    ACCaLO, F       ; negate ACCa
0018 02B3         00149          incf    ACCaLO, F
0019 0643         00150          btfsc   STATUS,Z
001A 00F5         00151          decf    ACCaHI, F
001B 0275         00152          comf    ACCaHI, F
001C 0800         00153          retlw   00
```


© 1997 Microchip Technology Inc.

```
                    00154
                    00155 ; *****************************************
                    00156 ; divide by 16 and limit to 100 Decimal
                    00157
                    00158 SHIFT   MACRO
                    00159         BCF     SWR,CARRY
                    00160         RRF     L_byte, F
                    00161         BCF     SWR,CARRY
                    00162         RRF     H_byte, F
                    00163         BTFSC   SWR,CARRY
                    00164         BSF     L_byte,7
                    00165         ENDM
                    00166
001D                00167 DIV_LMT
                    00168         SHIFT
001D 0403               M         BCF     SWR,CARRY
001E 0336               M         RRF     L_byte, F
001F 0403               M         BCF     SWR,CARRY
0020 0335               M         RRF     H_byte, F
0021 0603               M         BTFSC   SWR,CARRY
0022 05F6               M         BSF     L_byte,7
                    00169         SHIFT
0023 0403               M         BCF     SWR,CARRY
0024 0336               M         RRF     L_byte, F
0025 0403               M         BCF     SWR,CARRY
0026 0335               M         RRF     H_byte, F
0027 0603               M         BTFSC   SWR,CARRY
0028 05F6               M         BSF     L_byte,7
                    00170         SHIFT
0029 0403               M         BCF     SWR,CARRY
002A 0336               M         RRF     L_byte, F
002B 0403               M         BCF     SWR,CARRY
002C 0335               M         RRF     H_byte, F
002D 0603               M         BTFSC   SWR,CARRY
002E 05F6               M         BSF     L_byte,7
                    00171         SHIFT
002F 0403               M         BCF     SWR,CARRY
0030 0336               M         RRF     L_byte, F
0031 0403               M         BCF     SWR,CARRY
0032 0335               M         RRF     H_byte, F
0033 0603               M         BTFSC   SWR,CARRY
0034 05F6               M         BSF     L_byte,7
                    00172
0035                00173 LMT100
0035 0C01           00174         MOVLW   1H              ; SUBTRACT 1 FROM THE HIGH BYTE TO SEE
0036 0095           00175         SUBWF   H_byte,0        ; IF THERE IS ANYTHING THERE, IF NOT,
0037 0703           00176         BTFSS   SWR,CARRY       ; THEN LEAVE THE LOW BYTE ALONE
0038 0A3C           00177         GOTO    L8_E            ; OTHERWISE GIVE THE LOW BYTE A FULL
0039 0C64           00178         MOVLW   64H             ; COUNT AND IT WILL HAVE BEEN LIMITED
003A 0036           00179         MOVWF   L_byte          ; TO 100
003B 0A42           00180         GOTO    LMT_EXIT
003C                00181 L8_E
003C 0C64           00182         MOVLW   64H             ; LIMIT THE MAGNITUDE OF THE VALUE TO
003D 0096           00183         SUBWF   L_byte,0        ; 100 DECIMAL
003E 0703           00184         BTFSS   SWR,CARRY
003F 0A42           00185         GOTO    LMT_EXIT
0040 0C64           00186         MOVLW   64H
0041 0036           00187         MOVWF   L_byte
0042                00188 LMT_EXIT
0042 0800           00189         RETLW   00
                    00190 ;
                    00191 ;THE ROUTINE CALCTIMES DOES THE FOLLOWING: PCNT = DUTY CYCLE IN %
                    00192 ; 100 - PCNT --> LO  AND PCNT --> HI. ZERO VALUES IN EITHER LO OR HI
                    00193 ;ARE FORCED TO 1.
0043                00194 CALCTIMES
0043 0209           00195         MOVF    PCNT,W          ; PUT REQUESTED % INTO W REGISTER
0044 0027           00196         MOVWF   HI              ; COPY ON MICROSECONDS IN TO HI TIME
0045 0C64           00197         MOVLW   64H
0046 0028           00198         MOVWF   LO
0047 0209           00199         MOVF    PCNT,0
0048 00A8           00200         SUBWF   LO,1            ; LEAVE 100-HI TIME IN LO TIME
0049 0207           00201         MOVF    HI,0            ; INSPECT THE HIGH TIME
004A 0643           00202         BTFSC   SWR,2           ; IF ITS IS ZERO
004B 02A7           00203         INCF    HI,1            ; INCREMENT IT
004C 0208           00204         MOVF    LO,0            ; INSPECT THE LO TIME
004D 0643           00205         BTFSC   SWR,2           ; IF ITS ZERO
004E 02A8           00206         INCF    LO,1            ; INCREMENT IT
004F 0800           00207         RETLW   00
                    00208
                    00209
                    00210 ;*****************************************************************
0050                00211 BEGIN
0050 0000           00212         NOP             ; STUBBED BEGINNING
```

```
                    00213
                    00214
                    00215 ;****CHECKING THE LIMIT SWITCHES AND CHECKING FOR MW***************
                    00216 ; This will check the switch inputs for closure and will terminate
                    00217 ; pulsing if one is closed.  It doesn't distinguish between the switches
                    00218 ; so they are not dedicated to cw end and ccw end.
                    00219
0051                00220 SW_TRAP
0051 0004           00221         CLRWDT
0052 0746           00222         BTFSS   PORTB,2         ; THIS WILL TEST ALL THREE OF THE
0053 0A51           00223         GOTO    SW_TRAP         ; SWITCH INPUTS.  IF ANY ONE IS
0054 0766           00224         BTFSS   PORTB,3         ; SET THEN EXECUTION OF THE CODE
0055 0A51           00225         GOTO    SW_TRAP         ; WILL BE LIMITED TO LOOKING FOR
0056 0786           00226         BTFSS   PORTB,4         ; IT TO BE CLEARED
0057 0A51           00227         GOTO    SW_TRAP
                    00228
                    00229
                    00230 ;****RECEIVING THE POSITIONAL REQUEST*******************************
                    00231 ; The host system that wishes to send positional requests to the positioner
                    00232 ; servo makes its desire known by setting the chip select to the positioner
                    00233 ; low.  It then monitors the busy (Data Out) line from the positioner.  When
                    00234 ; the positioner sets the busy line high, the host may begin sending its 8 bit
                    00235 ; request.  The data bits should be valid on the rising edge of the clock.
                    00236 ; After 8 bits have been received by the positioner it will begin operation
                    00237 ; to send the system to the received position.  It can be interrupted at any
                    00238 ; point during the positioning process by the host sending a new command.  The
                    00239 ; opportunity to update the command is issued every 100 pwm pulses (every 50
                    00240 ; milliseconds).
                    00241 ; If the host sends a zero positional command the positioner will stop the
                    00242 ; system and remain inactive.
                    00243 ; If the host does not successfully complete a microwire transmission of 8
                    00244 ; data bits the watchdog timer will trip and reset the system to an inactive
                    00245 ; "stopped" state.
                    00246
                    00247
0058                00248 REC_MW
0058 0C0B           00249         MOVLW   0BH             ; RESET THE PORT FOR THREE INPUTS
0059 0005           00250         TRIS    PORTA           ; AND ONE OUTPUT
005A 0445           00251         BCF     PORTA,MWDO      ; SET THE DATA OUT LOW FOR BUSY
005B 0C20           00252         MOVLW   20H
005C 0037           00253         MOVWF   count
005D                00254 WATCH_CS
005D 0705           00255         BTFSS   PORTA,MWCS      ; CHECK FOR INCOMING REQUESTS
005E 0A62           00256         GOTO    REC_CMD         ; RECEIVE A NEW POSITION REQUEST
005F 02F7           00257         DECFSZ  count,1
0060 0A5D           00258         GOTO    WATCH_CS
0061 0A71           00259         GOTO    REC_EXIT        ; NO REQUEST WAS MADE IN THE TIME ALLOTED
0062                00260 REC_CMD
0062 0545           00261         BSF     PORTA,MWDO      ; SET THE DATA OUT HIGH FOR "OK TO SEND"
0063 0C08           00262         MOVLW   8H              ; SET TO RECEIVE 8 BITS
0064 0037           00263         MOVWF   count
0065                00264 WAIT_UP
0065 0765           00265         BTFSS   PORTA,MWCK      ; WAIT FOR A RISING EDGE
0066 0A65           00266         GOTO    WAIT_UP
0067 0403           00267         BCF     SWR,CARRY       ; RESET THE CARRY TO A DEFAULT ZERO
0068 0625           00268         BTFSC   PORTA,MWDI      ; READ THE DATA IN
0069 0503           00269         BSF     SWR,CARRY       ; SET THE CARRY FOR A ONE
006A 0370           00270         RLF     POSR,1          ; ROTATE THE BIT INTO THE POSITION REQ.
006B 02F7           00271         DECFSZ  count,1         ; DECREMENT THE BIT COUNTER
006C 0A6E           00272         GOTO    WAIT_DN         ; WAIT FOR THE FALLING EDGE
006D 0A71           00273         GOTO    REC_EXIT        ; LAST BIT RECEIVED
006E                00274 WAIT_DN
006E 0665           00275         BTFSC   PORTA,MWCK      ; CHECK THE INCOMING CLOCK
006F 0A6E           00276         GOTO    WAIT_DN         ; IF IT IS STILL HIGH WAIT FOR IT TO GO LOW
0070 0A65           00277         GOTO    WAIT_UP         ; IF IT GOES LOW GO BACK TO RECEIVE NEXT BIT
                    00278
0071                00279 REC_EXIT
0071 0445           00280         BCF     PORTA,MWDO      ; SET THE BUSY FLAG
                    00281
                    00282
                    00283 ;********** CHECK FOR THE DISABLE REQUEST *************************
                    00284 ; Position 0 is considered a request to not drive the system.  In this way
                    00285 ; the positioner will come up from a reset in a safe state and will not
                    00286 ; try to move the system to some arbitrary location.
                    00287
0072                00288 MOVE
0072 0210           00289         MOVF    POSR,W          ; CHECK THE REQUESTED POSTION
0073 0643           00290         BTFSC   SWR,Z           ; IF IT IS ZERO THEN WAIT FOR A NON-ZERO
0074 0A50           00291         GOTO    BEGIN           ; REQUEST BY BRANCHING BACK TO THE BEGINNING
                    00292
                    00293 ;****READING THE A/D VALUES****************************************
                    00294 ;
                    00295 ; Read the positional a/d channel (1) and store the value in the actual
```


© 1997 Microchip Technology Inc.

```
                00296 ; position variable (POSA).
                00297 ; This is written in line to minimize the use of variables
                00298
0075            00299 READ_POS
0075 0071       00300          CLRF    POSA            ; CLEAN THE POSITION ACTUAL HOLDER
0076 04E6       00301          BCF     PORTB,CSN       ; SET THE CHIP SELECT LOW TO A/D
0077 0C1C       00302          MOVLW   1CH             ; SET THE DATA LINE TO OUTPUT
0078 0006       00303          TRIS    PORTB           ; FOR SENDING SET-UP BITS
0079 05C6       00304          BSF     PORTB,BV        ; SET FOR "START" BIT
007A 0000       00305          NOP
                00306          CLKUP                   ; CLOCK IN THE START BIT
007B 05A6           M          BSF     PORTB,CK        ; data acquisition from the a/d
007C 0000           M          NOP
                00307          CLKDN                   ; "
007D 04A6           M          BCF     PORTB,CK        ; data acquisition from the a/d
007E 0000           M          NOP
                00308          CLKUP                   ; CLOCK IN SINGLE-ENDED
007F 05A6           M          BSF     PORTB,CK        ; data acquisition from the a/d
0080 0000           M          NOP
                00309          CLKDN                   ; "
0081 04A6           M          BCF     PORTB,CK        ; data acquisition from the a/d
0082 0000           M          NOP
                00310          CLKUP                   ; CLOCK IN CHANNEL 1
0083 05A6           M          BSF     PORTB,CK        ; data acquisition from the a/d
0084 0000           M          NOP
                00311          CLKDN                   ; TO THE MUX
0085 04A6           M          BCF     PORTB,CK        ; data acquisition from the a/d
0086 0000           M          NOP
0087 0C5C       00312          MOVLW   5CH             ; SET THE DATA LINE TO INPUT
0088 0006       00313          TRIS    PORTB           ; TO RECEIVE DATA BITS FROM A/D
                00314          CLKUP                   ; CLOCK UP TO LET MUX SETTLE
0089 05A6           M          BSF     PORTB,CK        ; data acquisition from the a/d
008A 0000           M          NOP
                00315          CLKDN                   ; CLOCK DN TO LET MUX SETTLE
008B 04A6           M          BCF     PORTB,CK        ; data acquisition from the a/d
008C 0000           M          NOP
                00316          GET_BIT                 ; GET BIT 7
008D 0403           M          BCF     SWR,CARRY
008E 05A6           M          BSF     PORTB,CK        ; SET CLOCK BIT HIGH
008F 06C6           M          BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
0090 0503           M          BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
0091 0371           M          RLF     POSA, F         ; ROTATE THE W REG LEFT
0092 04A6           M          BCF     PORTB,CK        ; SET THE CLOCK LOW
0093 0000           M          NOP                     ; DELAY
                00317          GET_BIT                 ; BIT 6
0094 0403           M          BCF     SWR,CARRY
0095 05A6           M          BSF     PORTB,CK        ; SET CLOCK BIT HIGH
0096 06C6           M          BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
0097 0503           M          BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
0098 0371           M          RLF     POSA, F         ; ROTATE THE W REG LEFT
0099 04A6           M          BCF     PORTB,CK        ; SET THE CLOCK LOW
009A 0000           M          NOP                     ; DELAY
                00318          GET_BIT                 ; BIT 5
009B 0403           M          BCF     SWR,CARRY
009C 05A6           M          BSF     PORTB,CK        ; SET CLOCK BIT HIGH
009D 06C6           M          BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
009E 0503           M          BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
009F 0371           M          RLF     POSA, F         ; ROTATE THE W REG LEFT
00A0 04A6           M          BCF     PORTB,CK        ; SET THE CLOCK LOW
00A1 0000           M          NOP                     ; DELAY
                00319          GET_BIT                 ; BIT 4
00A2 0403           M          BCF     SWR,CARRY
00A3 05A6           M          BSF     PORTB,CK        ; SET CLOCK BIT HIGH
00A4 06C6           M          BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
00A5 0503           M          BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
00A6 0371           M          RLF     POSA, F         ; ROTATE THE W REG LEFT
00A7 04A6           M          BCF     PORTB,CK        ; SET THE CLOCK LOW
00A8 0000           M          NOP                     ; DELAY
                00320          GET_BIT                 ; BIT 3
00A9 0403           M          BCF     SWR,CARRY
00AA 05A6           M          BSF     PORTB,CK        ; SET CLOCK BIT HIGH
00AB 06C6           M          BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
00AC 0503           M          BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
00AD 0371           M          RLF     POSA, F         ; ROTATE THE W REG LEFT
00AE 04A6           M          BCF     PORTB,CK        ; SET THE CLOCK LOW
00AF 0000           M          NOP                     ; DELAY
                00321          GET_BIT                 ; BIT 2
00B0 0403           M          BCF     SWR,CARRY
00B1 05A6           M          BSF     PORTB,CK        ; SET CLOCK BIT HIGH
00B2 06C6           M          BTFSC   PORTB,BV        ; LOOK AT DATA COMMING IN
00B3 0503           M          BSF     SWR,CARRY       ; SET THE CARRY FOR A 1
00B4 0371           M          RLF     POSA, F         ; ROTATE THE W REG LEFT
00B5 04A6           M          BCF     PORTB,CK        ; SET THE CLOCK LOW
```

```
00B6 0000              M         NOP                    ; DELAY
                    00322        GET_BIT                ; BIT 1
00B7 0403              M         BCF     SWR,CARRY
00B8 05A6              M         BSF     PORTB,CK       ; SET CLOCK BIT HIGH
00B9 06C6              M         BTFSC   PORTB,BV       ; LOOK AT DATA COMMING IN
00BA 0503              M         BSF     SWR,CARRY      ; SET THE CARRY FOR A 1
00BB 0371              M         RLF     POSA, F        ; ROTATE THE W REG LEFT
00BC 04A6              M         BCF     PORTB,CK       ; SET THE CLOCK LOW
00BD 0000              M         NOP                    ; DELAY
                    00323        GET_BIT                ; BIT 0
00BE 0403              M         BCF     SWR,CARRY
00BF 05A6              M         BSF     PORTB,CK       ; SET CLOCK BIT HIGH
00C0 06C6              M         BTFSC   PORTB,BV       ; LOOK AT DATA COMMING IN
00C1 0503              M         BSF     SWR,CARRY      ; SET THE CARRY FOR A 1
00C2 0371              M         RLF     POSA, F        ; ROTATE THE W REG LEFT
00C3 04A6              M         BCF     PORTB,CK       ; SET THE CLOCK LOW
00C4 0000              M         NOP                    ; DELAY
00C5 05E6           00324        BSF     PORTB,CSN      ; DESELECT THE CHIP
                    00325
                    00326
                    00327 ;****************** CALCULATING THE PID TERMS ***********************
                    00328
                    00329 ;****CALCULATE THE ERROR*******
                    00330 ; The error is very simply the signed difference between where the
                    00331 ; system is and where it is supposed to be at a particular instant
                    00332 ; in time.  It is formed by subtracting the actual position from the
                    00333 ; requested position (Position requested - Position actual).  This
                    00334 ; difference is then used to determine the proportional,integral and
                    00335 ; differential term contributions to the output.
                    00336
00C6                00337 C_ERR
00C6 0211           00338         MOVF    POSA,0         ; LOAD THE ACTUAL POSITION INTO W
00C7 0090           00339         SUBWF   POSR,0         ; SUBTRACT IT FROM THE REQUESTED POSITION
00C8 0603           00340         BTFSC   SWR,CARRY      ; CHECK THE CARRY BIT TO DETERMINE THE SIGN
00C9 0ACB           00341         GOTO    PLS_ER         ; ITS POSITIVE(POSR>POSA)
00CA 0ACE           00342         GOTO    MNS_ER         ; ITS NEGATIVE  (POSA>POSR)
                    00343
00CB                00344 PLS_ER
00CB 002C           00345         MOVWF   ERR1           ; SAVE THE DIFFERENCE IN "ERROR"
00CC 0419           00346         BCF     FLAGS,ER_SGN   ; SET THE SIGN FLAG TO INDICATE POSITIVE
00CD 0AD2           00347         GOTO    CE_EXIT
                    00348
00CE                00349 MNS_ER
00CE 0210           00350         MOVF    POSR,0         ; RE-DO THE SUBTRACTION
00CF 0091           00351         SUBWF   POSA,0         ; ACTUAL - REQUESTED
00D0 002C           00352         MOVWF   ERR1           ; STORE THE DIFFERENCE IN "ERROR"
00D1 0519           00353         BSF     FLAGS,ER_SGN   ; SET THE SIGN FLAG FOR NEGATIVE
                    00354
00D2                00355 CE_EXIT
00D2 006D           00356         CLRF    SUMLO          ; CLEAN OLD VALUES OUT TO PREPARE
00D3 0078           00357         CLRF    SUMHI          ; FOR THIS CYCLES SUMMATION
                    00358
                    00359 ;****CALCULATE THE PROPORTIONAL TERM******
                    00360 ; The proportional term is the error times the proportional gain term.
                    00361 ; This term simply gives you more output drive the farther away you are
                    00362 ; from where you want to be (error)*Kp.
                    00363 ; The proportional gain term is a signed term between -100 and 100  The
                    00364 ; more proportional gain you have the lower your system following error
                    00365 ; will be.  The higher your proportional gain, the more integral and
                    00366 ; differential term gains you will have to add to make the system stable.
                    00367 ; The sum is being carried as a 16 bit signed value.
                    00368
00D4                00369 C_PROP
00D4 020C           00370         MOVF    ERR1,0         ; LOAD THE ERROR TERM INTO W
00D5 0033           00371         MOVWF   mulcnd         ; MULTIPLY IT BY THE PROPORTIONAL GAIN
00D6 0C30           00372         MOVLW   KP             ; KP AND THEN SCALE IT DOWN BY DIVIDING
00D7 0034           00373         MOVWF   mulplr         ; IT DOWN BY 16. IF IT IS STILL OVER
00D8 0901           00374         CALL    mpy_S          ; 255 THEN LIMIT IT TO 255
00D9 091D           00375         CALL    DIV_LMT
                    00376
00DA                00377 RESTORE_SGN
00DA 0719           00378         BTFSS   FLAGS,ER_SGN   ; IF THE ERROR SIGN IS NEGATIVE THEN
00DB 0ADE           00379         GOTO    ADDPROP        ; PUT THE SIGN INTO THE LOW BYTE
00DC 0276           00380         COMF    L_byte,1
00DD 02B6           00381         INCF    L_byte,1
                    00382
00DE                00383 ADDPROP
00DE 0216           00384         MOVF    L_byte,W       ; SAVE THE PROPORTIONAL PART
00DF 01ED           00385         ADDWF   SUMLO,1        ; IN THE SUM
00E0 0603           00386         BTFSC   SWR,CARRY      ; IF THE ADDITION CARRIED OUT THEN
00E1 02B8           00387         INCF    SUMHI,1        ; INCREMENT THE HIGH BYTE
00E2 0C00           00388         MOVLW   0              ; THEN
00E3 06ED           00389         BTFSC   SUMLO,7        ; SIGN EXTEND TO THE UPPER
```


© 1997 Microchip Technology Inc.

```
00E4 0CFF          00390         MOVLW   0FF             ; BYTE
00E5 01F8          00391         ADDWF   SUMHI,1
                   00392
                   00393
                   00394
                   00395 ;****CALCULATE THE INTEGRAL TERM******
                   00396 ; The integral term is an accumulation of the error thus far.  Its purpose
                   00397 ; is to allow even a small error to effect a large change.  It does this
                   00398 ; by adding a small number into an accumulator each cycle through the program.
                   00399 ; Thusly even a small error that exists for a while will build up to a large
                   00400 ; enough number to effect an output sufficient to move the system.  The effect
                   00401 ; that this integral accumulator has is modulated by the integral gain term KI.
                   00402 ; The integral of the error over time is multiplied by KI and the result is its
                   00403 ; contribution to the final summation for determining the output value.  This
                   00404 ; term helps to insure the long-term accuracy of the system is good.  A certain
                   00405 ; amount is necessary for this purpose but too much will cause oscillations.
                   00406 ; The integral is bounded in magnitude for two purposes.  The first is so that
                   00407 ; it never rolls over and changes sign.  The second is that it may saturate on
                   00408 ; long moves forcing an excessively large overshoot to "de-integrate" the error
                   00409 ; accumulated during the first of the moves.
                   00410
00E6               00411 C_INT
00E6 020C          00412         MOVF    ERR1,W          ; MOVE THE ERROR INTO THE W REG
00E7 0643          00413         BTFSC   SWR,Z           ; AND CHECK TO SEE IF IT IS ZERO
00E8 0AFF          00414         GOTO    ADDINT          ; IF SO THEN DONT CHANGE THE ACCUMULATOR
00E9 0619          00415         BTFSC   FLAGS,ER_SGN    ; TEST THE FLAGS TO FIND THE POLARITY
00EA 0AEE          00416         GOTO    MNS_1           ; OF THE ERROR .. 0 POSITIVE 1 NEGATIVE
00EB               00417 PLS_1
00EB 0C02          00418         MOVLW   KI              ; IF POSITIVE ADD ONE TO
00EC 01EE          00419         ADDWF   ACCUM,1         ; THE ERROR ACCUMULATOR
00ED 0AF0          00420         GOTO    LMTACM          ; THEN LIMIT IT TO +/-100
00EE               00421 MNS_1
00EE 0C02          00422         MOVLW   KI              ; IF NEGATIVE THEN SUBTRACT ONE
00EF 00AE          00423         SUBWF   ACCUM,1         ; FROM THE ERROR ACCUMULATOR
00F0               00424 LMTACM
00F0 06EE          00425         BTFSC   ACCUM,7         ; CHECK THE SIGN BIT OF THE ERROR ACCUMULATOR
00F1 0AF9          00426         GOTO    M_LMT           ; AND DO A POSITIVE OR NEGATIVE LIMIT
00F2               00427 P_LMT
00F2 0C9C          00428         MOVLW   9CH             ; FOR THE POSITIVE LIMIT ADD 156 TO THE
00F3 01CE          00429         ADDWF   ACCUM,0         ; NUMBER AND SEE IF YOU GENERATE A CARRY
00F4 0703          00430         BTFSS   SWR,CARRY       ; BY CHECKING THE CARRY FLAG
00F5 0AFF          00431         GOTO    ADDINT          ; IF NOT THEN ITS O.K.
00F6 0C64          00432         MOVLW   64H             ; IF SO THEN FORCE THE ACCUMULATOR TO
00F7 002E          00433         MOVWF   ACCUM           ; 100 DECIMAL
00F8 0AFF          00434         GOTO    ADDINT
00F9               00435 M_LMT
00F9 0C9C          00436         MOVLW   9CH             ; FOR THE NEGATIVE LIMIT SUBTRACT 156 FROM
00FA 008E          00437         SUBWF   ACCUM,0         ; THE NUMBER AND SEE IF YOU GENERATE A
00FB 0603          00438         BTFSC   SWR,CARRY       ; NON-CARRY CONDITION INDICATING A ROLL-OVER
00FC 0AFF          00439         GOTO    ADDINT          ; IF NOT THEN LEAVE THE ACCUMULATOR ALONE
00FD 0C9C          00440         MOVLW   9CH             ; IF SO THEN LIMIT IT TO -100 BY
00FE 002E          00441         MOVWF   ACCUM           ; FORCING THAT VALUE IN THE ACCUMULATOR
                   00442
00FF               00443 ADDINT
00FF 020E          00444         MOVF    ACCUM,W         ; ADD THE INTEGRAL ACCUMULATOR TO
0100 01ED          00445         ADDWF   SUMLO,1         ; THE LOW BYTE OF THE SUM
0101 0603          00446         BTFSC   SWR,CARRY       ; TEST FOR OVERFLOW, IF SO THEN
0102 02B8          00447         INCF    SUMHI,1         ; INCREMENT THE HI BYTE
0103 0C00          00448         MOVLW   0               ; LOAD 0 INTO THE W REGISTER
0104 06EE          00449         BTFSC   ACCUM,7         ; IF THE INTEGRAL ACCUMULATOR WAS NEGATIVE
0105 0240          00450         COMF    W,W             ; COMPLEMENT THE 0 TO GET SIGN FOR HIGH BYTE
0106 01F8          00451         ADDWF   SUMHI,1         ; ADD INTO THE HIGH BYTE OF THE SUM
                   00452
                   00453
0107               00454 U_DEXIT                        ; EXIT POINT FOR THE UP/DOWN CONTROL OF ACCUM
                   00455
                   00456
                   00457
                   00458 ;****CALCULATING THE DIFFERENTIAL TERM*************************
                   00459 ; The differential term examines the error and determines how much
                   00460 ; it has changed since the last cycle.  It does this by subtracting the
                   00461 ; old error from the new error.  Since the cycle time is relatively fixed
                   00462 ; we can use it as the "dt" of the desired "de/dt".  This derivative of the
                   00463 ; error is then multiplied by the differential gain term KD and becomes the
                   00464 ; differential term contribution for the final summation.
                   00465
                   00466 ; First, create the "de" term by doing a signed subtaction of new error
                   00467 ; minus the old error. (new_error - old_error)
                   00468
0107               00469 C_DIFF
0107 020C          00470         MOVF    ERR1,W          ; LOAD THE NEW ERROR INTO REGISTER
0108 0719          00471         BTFSS   FLAGS,ER_SGN
0109 0B0D          00472         GOTO    LO_BYTE
```

```
010A 026C  00473          COMF    ERR1,1          ; CORRECT THE VALUE TO BE 16 BIT
010B 028C  00474          INCF    ERR1,W
010C 026C  00475          COMF    ERR1,1          ; RESTORE IT FOR FUTURE USE TO 8 BIT MAGNITUDE
010D       00476 LO_BYTE
010D 0034  00477          MOVWF   ACCbLO          ; FOR SUBTRACTION
010E 0C00  00478          MOVLW   00
010F 0619  00479          BTFSC   FLAGS,ER_SGN    ; SIGN EXTEND THE UPPER BYTE
0110 0CFF  00480          MOVLW   0FF
0111 0036  00481          MOVWF   ACCbHI
0112 020F  00482          MOVF    ERR_O,W         ; LOAD THE OLD ERROR INTO OTHER REGISTER
0113 0799  00483          BTFSS   FLAGS,OER_SGN
0114 0B17  00484          GOTO    LO_BYTEO
0115 026F  00485          COMF    ERR_O,1         ; CORRECT THE VALUE TO BE 16 BIT
0116 028F  00486          INCF    ERR_O,W
0117       00487 LO_BYTEO
0117 0033  00488          MOVWF   ACCaLO          ; FOR SUBTRACTION
0118 0C00  00489          MOVLW   00
0119 0699  00490          BTFSC   FLAGS,OER_SGN   ; SIGN EXTEND THE UPPER BYTE
011A 0CFF  00491          MOVLW   0FF
011B 0035  00492          MOVWF   ACCaHI
011C 090F  00493          CALL    D_sub           ; PERFORM THE SUBTRACTION
           00494
011D       00495 STRIP_SGN
011D 06F6  00496          BTFSC   ACCbHI,7        ; TEST THE SIGN OF THE RESULT
011E 0B20  00497          GOTO    NEG_ABS
011F 0B25  00498          GOTO    POS_ABS
0120       00499 NEG_ABS
0120 0559  00500          BSF     FLAGS,DE_SGN    ; ITS NEGATIVE SO SET THE FLAG AND
0121 0274  00501          COMF    ACCbLO,1        ; COMPLEMENT THE VALUE
0122 0294  00502          INCF    ACCbLO,W
0123 002F  00503          MOVWF   ERR_O
0124 0B28  00504          GOTO    MULT_KD
0125       00505 POS_ABS
0125 0459  00506          BCF     FLAGS,DE_SGN    ; ITS POSITIVE SO SET RESET THE FLAG
0126 0214  00507          MOVF    ACCbLO,W        ; AND SAVE THE VALUE
0127 002F  00508          MOVWF   ERR_O
           00509
           00510 ; Then multiply by Kd
           00511
0128       00512 MULT_KD
0128 020F  00513          MOVF    ERR_O,W
0129 0033  00514          MOVWF   mulcnd          ; MOVE THE DE/DT TERM INTO THE MULCND REG.
012A 0C20  00515          MOVLW   KD              ; MOVE THE DIFFERENTIAL GAIN TERM INTO
012B 0034  00516          MOVWF   mulplr          ; MULPLR TO MULTIPLY THE DE/DT
012C 0901  00517          CALL    mpy_S           ; DO THE MULTIPLICATION
012D 091D  00518          CALL    DIV_LMT         ; SCALE AND LIMIT TO 100
           00519
012E       00520 RE_SGN
012E 0759  00521          BTFSS   FLAGS,DE_SGN    ; IF THE DE SIGN IS NEGATIVE THEN
012F 0B32  00522          GOTO    SAVE_DIFF       ; PUT THE SIGN INTO THE LOW BYTE
0130 0276  00523          COMF    L_byte,1
0131 02B6  00524          INCF    L_byte,1
0132       00525 SAVE_DIFF
0132 0216  00526          MOVF    L_byte,W
0133 0643  00527          BTFSC   SWR,Z
0134 0B45  00528          GOTO    ROLL_ER
0135 002F  00529          MOVWF   ERR_O
           00530
           00531 ; ADD THE DIFF TERM INTO THE SUMM ***************
           00532
0136       00533 ADDDIF
0136 0C00  00534          MOVLW   00
0137 0659  00535          BTFSC   FLAGS,DE_SGN    ; PUT THE KD*(DE/DT) TERM INTO THE
0138 0CFF  00536          MOVLW   0FF             ; REGISTERS TO ADD. AND
0139 0036  00537          MOVWF   ACCbHI          ; SIGN EXTEND THE UPPER BYTE
013A 020F  00538          MOVF    ERR_O,W
013B 0034  00539          MOVWF   ACCbLO
013C 020D  00540          MOVF    SUMLO,W         ; LOAD THE CURRENT SUM INTO THE
013D 0033  00541          MOVWF   ACCaLO          ; REGISTERS TO ADD
013E 0218  00542          MOVF    SUMHI,W
013F 0035  00543          MOVWF   ACCaHI
0140 0910  00544          CALL    D_add           ; ADD IN THE DIFFERENTIAL TERM
0141 0214  00545          MOVF    ACCbLO,W        ; SAVE THE RESULTS BACK
0142 002D  00546          MOVWF   SUMLO           ; INTO SUMLO AND HI
0143 0216  00547          MOVF    ACCbHI,W
0144 0038  00548          MOVWF   SUMHI
           00549
0145       00550 ROLL_ER
0145 020C  00551          MOVF    ERR1,W          ; TAKE THE CURRENT ERROR
0146 002F  00552          MOVWF   ERR_O           ; AND PUT IT IN THE ERROR HISTORY
0147 0499  00553          BCF     FLAGS,OER_SGN   ; SAVE THE CURRENT ERROR SIGN
0148 0619  00554          BTFSC   FLAGS,ER_SGN    ; IN THE OLD ERROR SIGN FOR
0149 0599  00555          BSF     FLAGS,OER_SGN   ; NEXT TIME THROUGH
```


© 1997 Microchip Technology Inc.

```
                        00556
                        00557
                        00558 ;****SET UP THE DIRECTION FOR THE BRIDGE*********************
                        00559 ;
                        00560 ; After the sum of all the components has been made, the sign of the
                        00561 ; sum will determine which way the bridge should be powered.
                        00562 ; If the sum is negative the bridge needs to be set to drive ccw; if the
                        00563 ; sum is Positive then the bridge needs to be set to drive cw.  This
                        00564 ; is purely a convention and depends upon the polarity the motor and feedback
                        00565 ; element are hooked up in.
                        00566
014A                    00567 SET_DIR
014A 0479               00568         BCF     FLAGS,DIR       ; SET FOR DEFAULT CLOCKWISE
014B 06F8               00569         BTFSC   SUMHI,7         ; LOOK AT THE SIGN BIT, IF IT IS SET
014C 0579               00570         BSF     FLAGS,DIR       ; THEN SET FOR CCW BRIDGE DRIVE
                        00571
                        00572
                        00573 ;**** SCALE THE NUMBER TO BETWEEN 0 AND 100% **********************
                        00574
                        00575 ; After the direction is set the request for duty cycle is limited to between
                        00576 ; 0 and 100 percent inclusive.  This value is passed to the dutycycle setting
                        00577 ; routine by loading it in the variable "PCNT".
                        00578
                        00579
014D                    00580 L_SUMM
014D 07F8               00581         BTFSS   SUMHI,7         ; CHECK TO SEE IF IT IS NEGATIVE
014E 0B52               00582         GOTO    POS_LM
014F 0278               00583         COMF    SUMHI,1
0150 026D               00584         COMF    SUMLO,1
0151 02AD               00585         INCF    SUMLO,1
                        00586
0152                    00587 POS_LM
0152 0C01               00588         MOVLW   1H              ; SUBTRACT 1 FROM THE HIGH BYTE TO SEE
0153 0098               00589         SUBWF   SUMHI,0         ; IF THERE IS ANYTHING THERE, IF NOT,
0154 0703               00590         BTFSS   SWR,CARRY       ; THEN LEAVE THE LOW BYTE ALONE
0155 0B59               00591         GOTO    LB_L            ; OTHERWISE GIVE THE LOW BYTE A FULL
0156 0C64               00592         MOVLW   64H             ; COUNT AND IT WILL HAVE BEEN LIMITED
0157 002D               00593         MOVWF   SUMLO           ; TO 100
0158 0B5F               00594         GOTO    LP_EXIT         ; GOTO LIMIT PERCENT EXIT
0159                    00595 LB_L
0159 0C64               00596         MOVLW   64H             ; LIMIT THE MAGNITUDE OF THE VALUE TO
015A 008D               00597         SUBWF   SUMLO,0         ; 100 DECIMAL
015B 0703               00598         BTFSS   SWR,CARRY
015C 0B5F               00599         GOTO    LP_EXIT
015D 0C64               00600         MOVLW   64H
015E 002D               00601         MOVWF   SUMLO
                        00602
015F                    00603 LP_EXIT
015F 020D               00604         MOVF    SUMLO,W         ; STORE THE LIMITED VALUE IN
0160 0029               00605         MOVWF   PCNT            ; THE PERCENT DUTYCYCLE REQUEST
                        00606
                        00607
                        00608 ;*********************************************************
                        00609 ; PWM GENERATING ROUTINE
                        00610 ;
                        00611 ; The important thing here is not to have to do too many decisions or
                        00612 ; calculations while you are generating the 100 or so pulses.  These will
                        00613 ; take time and limit the minimum or maximum duty cycle.
                        00614
0161                    00615 WHICH_DIR
0161 0679               00616         BTFSC   FLAGS,DIR       ; CHECK THE DIRECTION FLAG
0162 0B76               00617         GOTO    GOCCW           ; DO CCW PULSES FOR 1
0163 0B64               00618         GOTO    GOCW            ; DO CW PULSES FOR 0
                        00619
                        00620
0164                    00621 GOCW
0164 0426               00622         BCF     PORTB,PWMCCW    ; SET THE BRIDGE FOR CW MOVE
0165 0C64               00623         MOVLW   64H             ;
0166 0032               00624         MOVWF   CYCLES          ; SET UP CYCLES COUNTER FOR 100 PULSES
0167 0943               00625         CALL    CALCTIMES       ; CALCULATE THE HI AND LO TIMES
                        00626
0168                    00627 RLDCW
0168 0207               00628         MOVF    HI,0            ; RELOAD THE HI TIMER
0169 002A               00629         MOVWF   HI_T            ; WITH THE CALCULATED TIME
016A 0208               00630         MOVF    LO,0            ; RELOAD THE LO TIMER
016B 002B               00631         MOVWF   LO_T            ; WITH THE CALCULATED TIME
016C 0004               00632         CLRWDT                  ; TAG THE WATCHDOG TIMER
                        00633
016D                    00634 CWHI
016D 0506               00635         BSF     PORTB,PWMCW     ; SET THE CLOCKWISE PWMBIT HIGH
016E 02EA               00636         DECFSZ  HI_T,1          ; DECREMENT THE HI USEC. COUNTER
016F 0B6D               00637         GOTO    CWHI            ; DO ANOTHER LOOP
0170                    00638 CWLO
```

```
0170 0406           00639          BCF     PORTB,PWMCW      ; SET THE CLOCKWISE PWM BIT LOW
0171 02EB           00640          DECFSZ  LO_T,1           ; DECREMENT THE LO USEC. COUNTER
0172 0B70           00641          GOTO    CWLO             ; DO ANOTHER LOOP
0173 02F2           00642          DECFSZ  CYCLES,1         ; DECREMENT THE NUMBER OF CYCLES LEFT
0174 0B68           00643          GOTO    RLDCW            ; DO ANOTHER PULSE
0175 0A50           00644          GOTO    BEGIN            ; DO ANOTHER MAIN SYSTEM CYCLE
                    00645
                    00646
0176                00647 GOCCW
0176 0406           00648          BCF     PORTB,PWMCW      ; SET THE BRIDGE FOR CCW MOVE
0177 0C64           00649          MOVLW   64H              ;
0178 0032           00650          MOVWF   CYCLES           ; SET UP CYCLE COUNTER FOR 100 PULSES
0179 0943           00651          CALL    CALCTIMES        ; CALCULATE THE HI AND LO TIMES
017A                00652 RLDCCW
017A 0207           00653          MOVF    HI,0             ; RE LOAD THE HI TIMER
017B 002A           00654          MOVWF   HI_T             ; WITH THE CALCULATED TIME
017C 0208           00655          MOVF    LO,0             ; RE LOAD THE LO TIMER
017D 002B           00656          MOVWF   LO_T             ; WITH THE CALCULATED TIME
017E 0004           00657          CLRWDT                   ; TAG THE WATCHDOG
                    00658
017F                00659 CCWHI
017F 0526           00660          BSF     PORTB,PWMCCW     ; SET THE COUNTERCLOCKWISE PWM BIT HIGH
0180 02EA           00661          DECFSZ  HI_T,1           ; DECREMENT THE HI USEC. COUNTER
0181 0B7F           00662          GOTO    CCWHI            ; DO ANOTHER LOOP
0182                00663 CCWLO
0182 0426           00664          BCF     PORTB,PWMCCW     ; SET THE COUNTERCLOCKWISE PWM BIT LOW
0183 02EB           00665          DECFSZ  LO_T,1           ; DECREMENT THE LO USEC. COUNTER
0184 0B82           00666          GOTO    CCWLO            ; DO ANOTHER LOOP
0185 02F2           00667          DECFSZ  CYCLES,1         ; DECREMENT THE NUMBER OF CYCLES LEFT
0186 0B7A           00668          GOTO    RLDCCW           ; DO ANOTHER PULSE
0187 0A50           00669          GOTO    BEGIN            ; DO ANOTHER MAIN SYSTEM CYCLE
                    00670
                    00671
                    00672
                    00673
                    00674
                    00675          ;************* START VECTOR ********************
                    00676
0188                00677 CLRREG                            ;INITIALIZE REGISTERS
                    00678
0188 0C0B           00679          MOVLW   0BH              ; SET PORT A FOR 3 INPUTS AND
0189 0005           00680          TRIS    PORTA            ; AN OUTPUT
018A 0C1C           00681          MOVLW   1CH              ; SET PORT B FOR INPUTS AND OUTPUTS
018B 0006           00682          TRIS    PORTB            ; THIS SETTING FOR SENDING TO A/D
018C 0040           00683          CLRW                     ; CLEAR THE W REGISTER
018D 0002           00684          OPTION                   ; STORE THE W REG IN THE OPTION REG
018E 0C08           00685          MOVLW   08H              ; STARTING REGISTER TO ZERO
018F 0024           00686          MOVWF   FSR              ;
0190                00687 GCLR
0190 0060           00688          CLRF    00               ;
0191 03E4           00689          INCFSZ  FSR, F           ; SKIP AFTER ALL REGISTERS
0192 0B90           00690          GOTO    GCLR             ; HAVE BEEN INITIALIZED
0193 0A50           00691          GOTO    BEGIN            ; START AT THE BEGINING OF THE PROGRAM
                    00692
01FF                00693          ORG     01FF             ;
01FF 0B88           00694          GOTO    CLRREG           ; START VECTOR
                    00695
                    00696
                    00697          END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXX------------ ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   405
Program Memory Words Free:   619


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Using a PIC16C5X as a Smart I²C™ Peripheral

Author: *Don Lekei*
NII Norsat International Inc.

INTRODUCTION

PIC16C5X microcontrollers from Microchip are ideally suited for use as smart peripheral devices under the control of the main processors in systems, due to their low cost and high speed. They are capable of performing tasks which would simply overload a conventional microprocessor, or require considerable logic circuitry, at a cost competitive with lower mid-range PLDs. To minimize the engineering overhead of adding multiple controllers to a product, it is convenient for the auxiliary controllers to emulate standard I/O peripherals.

A common interface found in existing products is the I²C bus. This efficient, two-wire, bi-directional interface allows the designer to connect multiple devices together, with the microprocessor able to send data to and receive data from any device on the bus. This interface is found on a variety of components, such as PLLs, DACs, video controllers, and EEPROMs. If a product already contains one or more I²C devices, it is simple to add a PIC16C5X emulating a compatible component.

This application note describes the implementation of a standard slave device with multiple, bi-directional registers. A subset of the full I²C specification is supported, which can be controlled by the same software which would talk to a Microchip 24LCXX series EEPROM.

THE I²C BUS

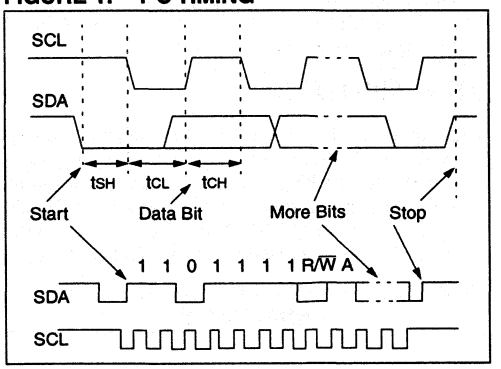
The I²C bus is a master-slave two-wire interface, consisting of a clock line (SCL) and a data line (SDA). Bi-directional communication (and in a full, multi-master system, collision detection, and clock synchronization) is facilitated through the use of a "wire-and" (i.e., active-low, passive-high) connection.

The standard mode I²C bus supports SCL clock frequencies up to 100 kHz. The fast-mode I²C bus supports clock rates up to 400 kHz. This application note will support the 100 kHz (standard-mode) clock rate.

Each device has a unique seven bit address, which the master uses to access each individual slave device.

During normal communication, the SDA line is only permitted to change while the SCL line is low, thus providing two violation conditions (Figure 1) which are used to signal a start condition (SDA drops while SCL is high) and a stop condition (SDA rises while SCL is high), which frame a message.

FIGURE 1: I²C TIMING



Each byte of a transfer is 9-bits long (see timing chart in the program listing). The talker sends 8 data bits followed by a '1' bit. The listener acknowledges the receipt of the byte and gives permission to send the next byte by inserting a '0' bit over the trailing '1'. The listener may indicate "not ready for data" by leaving the acknowledge bit as a '1'.

The clock is generated by the master only. The slave device must respond to the master within the timing specifications of the I²C definition otherwise the master would be required to operate in slow mode, which most software implementations of I²C masters do not actually support. The specified (standard-mode) t_{CL} is 4.7 µs, and t_{CH} is only 4 µs, so it would be extremely difficult to achieve the timing of a hardware slave device with a conventional microcontroller.

MESSAGE FORMAT

A message is always initiated by the master, and begins with a start condition, followed by a slave address (7 MSbs) and direction bit (LSb = '1' for READ, '0' for WRITE). The addressed slave must acknowledge this byte if it is ready to communicate any data. If the slave fails to respond, the master should initiate a stop condition and retry.

If the direction bit is '0' the next byte is considered the sub-address (this is an extension to I²C used by most multi-register devices). The sub-address selects which "register" or "function" subsequent read or write operations will affect. Any additional bytes will be received and stored in consecutive locations until a stop is sent. If the slave is unable to process more data, it could terminate transfer by not acknowledging the last byte.

© 1997 Microchip Technology Inc.

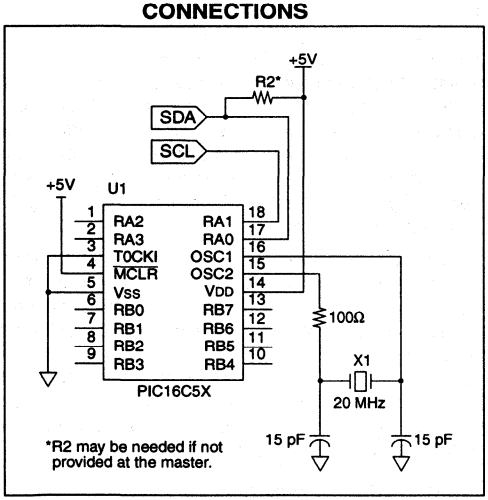
If the direction bit is '1', the slave will transfer successive bytes to the master (the master holds the line at '1'), while the master acknowledges each byte with a '0' in the ninth bit. The master can terminate the transfer by not acknowledging the last byte, while the slave can stop the transfer by generating a stop condition.

The start address of a read operation is set by sending a write request with a sub-address only (no data bytes). For a detailed set of timing diagrams and different communication modes, consult any of the Microchip 24LCXX EEPROM specifications. This program communicates using the same formats.

IMPLEMENTATION

The chip will respond to slave address "DEVICE_ADDRESS", which by default is $D6_{16}$ ($D7_{16}$ for read). This address was chosen because it is the fourth optional address of a Philips PCF8573 clock /calender or a TDA8443 tipple video switch (unlikely that a product would contain four of those).

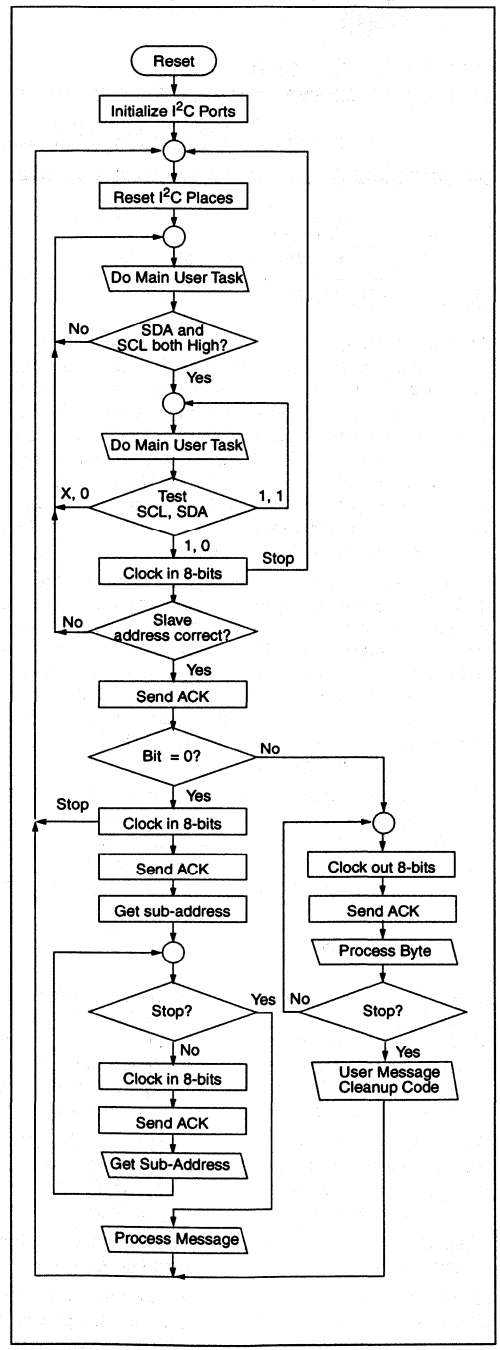
FIGURE 2: SCHEMATIC OF I²C CONNECTIONS



The connections to the device are shown in Figure 2. The use of RA0 for data input is required. Data is shifted directly out of the port. The code could be modified to make it port independent, but the loss of efficiency may hinder some real-time applications.

This application emulates an I²C device with 8 registers, accessed as sub-addresses 1 through 8 (module 7), plus a data channel (0). The example code returns an ID string when the data channel is accessed. When bytes are written to sub-addresses other than 0, they are stored in I2CR0-I2CR7 (I2CR0 gets data written to sub-address 8).

FIGURE 3: I²C DEVICE FLOWCHART



© 1997 Microchip Technology Inc.

When the initial subaddress is 0, the flag B_ID is set. This is used to indicate access to a special channel. In this case, the data channel is used to return an ID message, or output data to PORTB, however the natural extension would be to use this as a data I/O channel.

To make the basic device routines easily adaptable to a variety of uses, macros are used to implement the application specific code. This allows the developer the option of using subroutine calls, or in-line code to avoid the 4 clock cycle overhead and use of the precious stack.

Macro	User Code Function
USER_MAIN	Code to execute in the main loop while not in a message. If this code takes too long, tSH of 4 µs will be violated (Figure 1). The slave will simply miss the address, not acknowledge, and the master will retry.
USER_Q	This would be quick user code to implement real-time processes. In most applications, this macro would be empty. If used, this routine should be kept under 4 µs if possible.
USER_MSG	This would be user code to process a message. It is inserted after a message is successfully received.
USER_RECV	This would be user code to process a received byte. It allows the user to add extra code to implement special purpose sub-addresses such as FIFOs.
USER_XMIT	This would be user code to prepare an output byte. In the default routine, it traps sub-address 0 and calls the ID string function.

References:

I²C Bus Specification, Philips Corporation, December 1988.

The I²C bus and how to use it (including specification), Signetics/Philips Semiconductors, January 1992.

Fenger, Carl, "The Inter-Integrated Circuit (I²C) Serial Bus: Theory and Practical Consideration", Application Note 168, Philips Components, December 1988.

"24C16 16K CMOS Serial Electrically Erasable PROM", Microchip Data Book (1992).

About the Author:

Don Lekei has been designing microprocessor based products over 14 years. He has developed many software and hardware products for a wide variety of applications. Mr. Lekei is Manager of Advanced Technologies at NII Norsat International Inc. at their Canadian headquarters in Surrey, British Columbia. Norsat designs and manufactures products to receive broadcast communications from satellites, terrestrial broadcasting systems and optical fibre. Norsat develops technologies and products for satellite entertainment television, broadcast music and data networks.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40 Released              I2C.ASM   1-16-1997  14:51:28         PAGE  1


LOC  OBJECT CODE  LINE SOURCE TEXT
  VALUE

                  00001         LIST    P=16C54, C=80, N=0, R=DEC
                  00002 ;
                  00003 ;*****************************************************************
                  00004 ;
                  00005 ;       Program:          I2C.ASM
                  00006 ;       Revision Date:
                  00007 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                  00008 ;
                  00009 ;*****************************************************************
                  00010 ;
  00000676        00011 CPU     EQU     1654
  00000000        00012 SIM     EQU     0               ;Change timing constants for simulator
                  00013
                  00014         IF      (CPU==1654) || (CPU==1655)
  000001FF        00015 _RESVEC EQU     01FFH           ;16c54 start address
                  00016         ENDIF
                  00017
                  00018         IF      CPU==1656
                  00019 _RESVEC EQU     03FFH           ;16C56 start address
                  00020         ENDIF
                  00021
                  00022         IF      CPU==1657
                  00023 _RESVEC EQU     07FFH           ;16C57 start address
                  00024         ENDIF
                  00025
                  00026 ;*** Reset Vector ****************************************
                  00027
01FF              00028         ORG     _RESVEC         ;
01FF              00029 RESVEC                          ;
01FF 0A0B         00030         GOTO    INIT            ;
                  00031
                  00032 ;*********************************************************
                  00033
                  00034 ;*********************************************************
                  00035 ;* Macros to set/clear/branch/skip on bits
                  00036 ;* These macros define and use synthetic "bit labels"
                  00037 ;* Bit labels contain the address and bit of a location
                  00038 ;*
                  00039 ;*************************************************************************
                  00040 ;*      Usage                    Description
                  00041 ;*      ----------------------   --------------------
                  00042 ;*      BIT     label,bit,file   ;Define a bit label
                  00043 ;*      SEB     label            ;set bit using bit label
                  00044 ;*      CLB     label            ;clear bit using bit label
                  00045 ;*      SKBS    label            ;SKIP on bit set
                  00046 ;*      SKBC    label            ;SKIP on bit clear
                  00047 ;*      BBS     label,address    ;BRANCH on bit set
                  00048 ;*      BBC     label,address    ;BRANCH on bit clear
                  00049 ;*      CBS     label,address    ;CALL on bit set
                  00050 ;*      CBC     label,address    ;CALL on bit clear
                  00051 ;*
                  00052 ;*************************************************************************
                  00053
                  00054 BIT     MACRO   label,bit,file   ;Define a bit label
```


© 1997 Microchip Technology Inc.

```
                00055 label           EQU    file<<8|bit          ;(macro)
                00056         ENDM                               ;
                00057
                00058 SEB     MACRO   label                      ;Set bit
                00059                 BSF    label>>8,label &7   ;(macro)
                00060         ENDM                               ;
                00061
                00062 CLB     MACRO   label                      ;Clear bit
                00063                 BCF    label>>8,label &7   ;(macro)
                00064         ENDM                               ;
                00065
                00066 SKBS    MACRO   label                      ;Skip on bit set
                00067                 BTFSS  label>>8,label &7   ;(macro)
                00068         ENDM
                00069
                00070 SKBC    MACRO   label                      ;Skip on bit clear
                00071                 BTFSC  label>>8,label &7   ;(macro)
                00072         ENDM
                00073
                00074 BBS     MACRO   label,address              ;Branch on bit set
                00075                 BTFSC  label>>8,label &7   ;(macro)
                00076                 GOTO   address             ;(macro)
                00077         ENDM                               ;
                00078
                00079 BBC     MACRO   label,address              ;Branch on bit clear
                00080                 BTFSS  label>>8,label &7   ;(macro)
                00081                 GOTO   address             ;(macro)
                00082         ENDM
                00083
                00084 CBS     MACRO   label,address              ;Call on bit set
                00085                 CALL   label>>8,label  &7  ;(macro)
                00086         ENDM                   ;
                00087
                00088 CBC     MACRO   label,address              ;Call on bit clear
                00089                 CALL   label>>8,label   &7 ;(macro)
                00090         ENDM
                00091
                00092
                00093 ;For Assembler portability
                00094
  00000000      00095 W       EQU     0                          ;For file,W
  00000000      00096 w       EQU     0                          ;For file,W
  00000001      00097 F       EQU     1                          ;For file,F
  00000001      00098 f       EQU     1                          ;For file,F
                00099
                00100 ;*****************************************************************
                00101 ;* REGISTER DECLARATIONS
                00102 ;*****************************************************************
                00103
                00104
0000            00105         ORG     0                  ;ORG for register declarations
                00106
0000            00107 ind     RES     1                  ;0=pseudo-reg 0 for in direct (FSR)
0001            00108 TMR0    RES     1                  ;1=real time counter
0002            00109 PC      RES     1                  ;2=PC
0003            00110 STATUS  RES     1                  ;3=status reg
                00111
                00112 ;* Status reg bits
                00113
                00114         BIT     B_C,0,STATUS       ;Carry
  00000300          M B_C             EQU    STATUS<<8|0  ;(macro)
                00115         BIT     B_DC,1,STATUS      ;Half carry
  00000301          M B_DC             EQU    STATUS<<8|1 ;(macro)
                00116         BIT     B_Z,2,STATUS       ;Zero
  00000302          M B_Z             EQU    STATUS<<8|2  ;(macro)
                00117         BIT     B_PD,3,STATUS      ;Power down
```

```
  00000303        M B_PD          EQU     STATUS<<8|3  ;(macro)
                00118         BIT     B_TO,4,STATUS       ;Timeout
  00000304        M B_TO          EQU     STATUS<<8|4  ;(macro)
                00119         BIT     B_PA0,5,STATUS      ;Page select (56/57 only)
  00000305        M B_PA0         EQU     STATUS<<8|5  ;(macro)
                00120         BIT     B_PA1,6,STATUS      ;Page select (56/57 only)
  00000306        M B_PA1         EQU     STATUS<      ;(macro)
                00121         BIT     B_PA2,7,STATUS      ;GP flag
  00000307        M B_PA2         EQU     STATUS<<8|7  ;(macro)
                00122
0004            00123 FSR     RES     1                   ;4=file select reg 0-4 =indirect address
0005            00124 PORTA   RES     1                   ;5=port A I/O register (4 bits)
0006            00125 PORTB   RES     1                   ;6=port B I/O register
                00126
                00127         IF      (CPU==1655)||(CPU==1657)
                00128 PORTC   RES     1                   ;7=I/O port C on 16C54/56 only
                00129         ENDIF
                00130
                00131 ;registers used by this code
                00132
0007            00133 I2CFLG  RES     1                    ;I2C flag reg
                00134 ;--i2c flags-----------------------------------------------
                00135         BIT     B_RD,0,I2CFLG       ;Flag: 1=read
  00000700        M B_RD          EQU     I2CFLG<<8|0  ;(macro)
                00136         BIT     B_UA,1,I2CFLG       ;Flag: 0=reading unit address
  00000701        M B_UA          EQU     I2CFLG<<8|1  ;(macro)
                00137         BIT     B_SA,2,I2CFLG       ;Flag: 1=reading subabbress
  00000702        M B_SA          EQU     I2CFLG<<8|2  ;(macro)
                00138         BIT     B_ID,3,I2CFLG       ;Flag: 1=reading id
  00000703        M B_ID          EQU     I2CFLG<<8|3  ;(macro)
                00139 ;---------------------------------------------------------
0008            00140 I2CREG  RES     1                   ;I2C I/O register
0009            00141 I2CSUBA RES     1                   ;Subaddress
000A            00142 I2CBITS RES     1                   ;I2C xmit bit counter
                00143
                00144
                00145 ;*************************************************************************
                00146 ;* 8 Pseudo registers accessed by sub-addresses 1-8
                00147 ;* (address 0 accesses the ID string)
                00148 ;* these are read-write registers
                00149 ;*************************************************************************
                00150
                00151
  0000000B      00152 I2CR0   EQU     $                   ;Sub-address 8
000B            00153         RES     1                   ;8 pseudo registers
                00154
  0000000C      00155 I2CR1   EQU     $                   ;Sub-address 1
000C            00156         RES     1
                00157
  0000000D      00158 I2CR2   EQU     $                   ;Sub-address 2
000D            00159         RES     1
                00160
  0000000E      00161 I2CR3   EQU     $                   ;Sub-address 3
000E            00162         RES     1
                00163
  0000000F      00164 I2CR4   EQU     $                   ;Sub-address 4
000F            00165         RES     1
                00166
  00000010      00167 I2CR5   EQU     $                   ;Sub-address 5
0010            00168         RES     1
                00169
  00000011      00170 I2CR6   EQU     $                   ;Sub-address 6
0011            00171         RES     1
                00172
  00000012      00173 I2CR7   EQU     $                   ;Sub-address 7
0012            00174         RES     1
```


© 1997 Microchip Technology Inc.

```
                00175
                00176 ;Constants used by program
                00177
000000D6        00178 DEVICE_ADDRESS EQU  0D6H  ;I2C device address (device_address+1 = read)
                00179
                00180 ;*****************************************************************
                00181 ;** PORTA DEFINITIONS
                00182 ;** I2C interface uses PORTA
                00183 ;** note SDA goes to A0 for code efficiency
                00184 ;**
                00185 ;*****************************************************************
                00186
                00187
000000F7        00188 TAREAD  EQU     B'11110111'     ;TRISA register for SDA read
000000F6        00189 TAWRITE EQU     B'11110110'     ;TRISA register for SDA write
000000F7        00190 TAINIT  EQU     TAREAD          ;Initial TRISA value
                00191
                00192         BIT     B_SDA,0,PORTA   ;I2C SDA (data) This must be bit 0!
00000500            M B_SDA           EQU PORTA<<8|0  ;(macro)
                00193         BIT     B_SCL,1,PORTA   ;I2C SCL (clock)
00000501            M B_SCL           EQU PORTA<<8|1  ;(macro)
                00194 ;spare          B_???,2,PORTA   ;not used
                00195 ;spare          B_???,3,PORTA   ;not used
                00196
                00197 ;*****************************************************************
                00198 ;**
                00199 ;** Port B definition (Parallel out)
                00200 ;**
                00201 ;*****************************************************************
00000000        00202 TBINIT  EQU     B'00000000'     ;Port B tris (all output)
000000FF        00203 PBINIT  EQU     B'11111111'     ;Port B init
                00204
                00205
                00206 ;*************************************************************************
                00207 ;* Macros to contain user POLL loop code.
                00208 ;* These are implemented as macros to allow ease of modification,
                00209 ;* especially in real-time applications. The functions could be coded
                00210 ;* as in-line code or as subroutines depending on ROM/time tradeoffs.
                00211 ;*
                00212 ;* USER_MAIN:   Decision or code to perform at idle time
                00213 ;*
                00214 ;* USER_Q:      'Quick' code for use during transfer - max 8 Ês for
                00215 ;*             full I2C Spec. More than 4s may result in I2C
                00216 ;*             retries (at full spec speed.
                00217 ;*
                00218 ;* USER_MSG:    Code to execute at receipt of I2C command.
                00219 ;*
                00220 ;*************************************************************************
                00221
                00222 USER_MAIN       MACRO
                00223 ;*** This would be user code for idle loop
                00224         ENDM
                00225
                00226 USER_Q          MACRO
                00227 ;*** This would be quick user code
                00228         ENDM
                00229
                00230 USER_MSG        MACRO
                00231 ;*** This would be user code to process a message
                00232         ENDM
                00233
                00234 USER_RECV       MACRO
                00235 ;*** This would be user code to process a received byte
                00236 ;*** example code sends sub-address 0 to port b
                00237    BBC     B_ID,_NXI_notid ;Channel 0! Bit set if INITIAL address was 0
                00238    MOVFW   I2CREG          ;get received byte
```

```
                00239    MOVWF   PORTB           ;and write it on portb
                00240         GOTO    IN_CONT
                00241 _NXI_notid
                00242    ENDM
                00243
                00244 USER_XMIT       MACRO
                00245 ;*** This would be user code to prepare an output byte
                00246 ;*** example code sends id string to output
                00247    BBC     B_ID,_NXO_notid ;Channel 0! Bit set if INITIAL address was 0
                00248    CALL    GETID           ;get next byte from ID channel
                00249    GOTO    OUT_CONT        ;and send it
                00250 _NXO_notid
                00251    ENDM
                00252
                00253 ;*********************************************************************
                00254 ; START OF CODE
                00255 ;*********************************************************************
0000            00256         ORG     0
                00257 ;*********************************************************************
                00258 ;* Device ID Table (must be at start)
                00259 ;* TABLE FOR UNIT ID returns next char in W
                00260 ;*********************************************************************
0000            00261 GETID
0000 0209       00262     MOVFW   I2CSUBA         ;W=I2CSUBA
0001 0E07       00263     ANDLW   07H             ;Limit to 8 locations
0002 01E2       00264     ADDWF   PC,F
                00265
                00266 ;*********************************************************************
                00267 ;* Device ID text: read starting at sub-address 0
                00268 ;*********************************************************************
                00269
0003 0850       00270     RETLW   'P'
0004 0849       00271     RETLW   'I'
0005 0843       00272     RETLW   'C'
0006 0849       00273     RETLW   'I'
0007 0832       00274     RETLW   '2'
0008 0843       00275     RETLW   'C'
0009 0800       00276     RETLW   0
000A 0800       00277     RETLW   0
                00278
                00279
                00280 ;*********************************************************************
                00281 ;* I2C Device routines
                00282 ;*
                00283 ;* Enable must be HIGH, else state goes to 0
                00284 ;* write is to me, read is from me.
                00285 ;*
                00286 ;*       <============== first byte / subsequent writes =====> <end>
                00287 ;*
                00288 ;* SDA    --|X-----X-----X-----X-----X-----X-----X-----X---X------|  |--
                00289 ;*       |---X-----X-----X-----X-----X-----X-----X-----X---X******|--|
                00290 ;* (bit)    s     7     6     5     4     3     2     1     0   ackout
                00291 ;* SCL  -----|  |--|  |--|  |--|  |--|  |--|  |--|  |--|  |--|  |--|
                00292 ;*           |--|  |--|  |--|  |--|  |--|  |--|--|  |--|  |--|
                00293 ;*
                00294 ;* STATE: 0 1 2  3  2  3  2  3  2  3  2  3  2  3  2  3  2  4  5  6  2  3 0
                00295 ;*
                00296 ;*             <============== subsequant reads ====================>
                00297 ;*
                00298 ;* SDA      X-----X-----X-----X-----X-----X-----X-----X-----X----X------X-----
                00299 ;*        --X-----X-----X-----X-----X-----X-----X-----X-----X----X******X-----
                00300 ;* (bit)ack   7     6     5     4     3     2     1     0   ackin
                00301 ;* SCL  --|  |--|  |--|  |--|  |--|  |--|  |--|  |--|  |--|  |--|  |--|  |--
                00302 ;*         |--|  |--|  |--|  |--|  |--|  |--|  |--|  |--|  |--|  |--|
                00303 ;*
                00304 ;* STATE:  7  8  7  8  7  8  7  8  7  8  7  8  7  8  7  8  9  A  7  8
```


© 1997 Microchip Technology Inc.

```
                00305 ;*
                00306 ;*            <============== Final READ ========================>
                00307 ;*
                00308 ;* SDA     X-----X-----X-----X-----X-----X-----X-----X----X-|**|-------
                00309 ;*       --X-----X-----X-----X-----X-----X-----X-----X----X*|--|
                00310 ;* (bit)ack   7     6     5     4     3     2     1   0   ackin
                00311 ;* SCL  --|  |--|  |--|  |--|  |--|  |--|  |--|  |--|  |--|  |---------
                00312 ;*         |--|  |--|  |--|  |--|  |--|  |--|  |--|  |--|  |--|
                00313 ;*
                00314 ;* STATE:  7  8  7  8  7  8  7  8  7  8  7  8  7  8  7  8  9  A  0  0
                00315 ;*
                00316 ;* STATE B is an ignore bit state for non-addressed bits
                00317 ;* STATE C indicates last sample had ENA low on rising
                00318 ;*  edge of ENA, DATA LOW = low voltage, DATA&CLOCK low = RESET
                00319 ;*****************************************************************
                00320
                00321 ;I2C interface uses PORTA
                00322 ;note SDA must be on PORTA,0 for code efficiency
                00323
                00324 ;*****************************************************************
                00325 ;** INIT
                00326 ;** Hardware reset entry point
                00327 ;**
                00328 ;*****************************************************************
000B            00329 INIT    ;Power-on entry
                00330
                00331 ;*****************************************************************
                00332 ;** RESET
                00333 ;** software reset entry point
                00334 ;**
                00335 ;*****************************************************************
000B            00336 RESET   ;Soft reset
                00337
000B 0CF7       00338         MOVLW   TAINIT          ;Init ports
000C 0005       00339         TRIS    PORTA
000D 0C00       00340         MOVLW   TBINIT
000E 0006       00341         TRIS    PORTB
000F 0CFF       00342         MOVLW   PBINIT
0010 0026       00343         MOVWF   PORTB
                00344
                00345 ;*****************************************************************
                00346 ; Main wait loop while idle. POLL loop should be called here
                00347 ;
                00348 ;*****************************************************************
                00349
0011            00350 I2CWAIT
0011 0004       00351         CLRWDT                  ;Clear watchdog timer
                00352         CLB     B_UA            ;Init state flags
0012 0427           M         BCF     B_UA>>8,B_UA&7  ;(macro)
                00353         CLB     B_SA            ;Init state flags
0013 0447           M         BCF     B_SA>>8,B_SA&7  ;(macro)
                00354         CLB     B_RD            ;Init state flags
0014 0407           M         BCF     B_RD>>8,B_RD&7  ;(macro)
                00355
0015            00356 loop1
0015 0004       00357         CLRWDT                  ;Clear watchdog timer
                00358
                00359         USER_MAIN       ;Call user code while in idle state
                    M ;*** This would be user code for idle loop
                00360
                00361         SKBC    B_SDA           ;Wait for SDA&SCL=H
0016 0605           M         BTFSC   B_SDA>>8,B_SDA&7 ;(macro)
0017            00362 loop2
                00363         SKBS    B_SCL             ;
0017 0725           M         BTFSS   B_SCL>>8,B_SCL&7  ;(macro)
0018 0A15       00364         GOTO    loop1             ; No longer valid to wait for start!
```

```
                    00365
0019 0004           00366           CLRWDT                      ;Clear watchdog timer
                    00367
                    00368           USER_MAIN       ;Call user code while in idle state
                        M ;*** This would be user code for idle loop
                    00369
                    00370 ;** wait for start **
                    00371           SKBC    B_SCL               ;Clock has dropped
001A 0625               M           BTFSC   B_SCL>>8,B_SCL&7    ;(macro)
                    00372           SKBC    B_SDA               ;Data dropped... Start!
001B 0605               M           BTFSC   B_SDA>>8,B_SDA&7    ;(macro)
001C 0A17           00373           GOTO    loop2
                    00374
                    00375 ;** START RECEIVED! --- wait for first bit!
001D                00376 loop3
                    00377           BBS     B_SDA,I2CWAIT       ;Data raised before clock dropped -- abort
001D 0605               M           BTFSC   B_SDA>>8,B_SDA&7    ;(macro)
001E 0A11               M           GOTO    I2CWAIT             ;(macro)
                    00378           BBS     B_SCL,loop3         ;Wait for clock low
001F 0625               M           BTFSC   B_SCL>>8,B_SCL&7    ;(macro)
0020 0A1D               M           GOTO    loop3               ;(macro)
                    00379
                    00380
0021                00381 NEXTBYTE
0021 0004           00382           CLRWDT              ;Clear watchdog timer
0022 0C01           00383           MOVLW   1           ;Init receive byte so bit falls off at end!
0023 0028           00384           MOVWF   I2CREG
                    00385
                    00386 ;*Shift bits! external poll may be executed during low clock cycle only!
                    00387 ;*ENABLE line is checked for loss of enable ONLY during HIGH CLOCK
                    00388
                    00389 ;*** CLOCK IS LOW -- DATA MAY CHANGE HERE
                    00390 ;*** We have at least 4 Ês before any change can occur
                    00391
0024                00392 loop4
                    00393           USER_Q
                        M ;*** This would be quick user code
0024                00394 loop4A
                    00395           BBC     B_SCL,loop4A        ;Wait for clock high
0024 0725               M           BTFSS   B_SCL>>8,B_SCL&7    ;(macro)
0025 0A24               M           GOTO    loop4A              ;(macro)
                    00396
                    00397 ;*** CLOCK IS HIGH -- SHIFT BIT - then watch for chang
                              e
0026 0305           00398           RRF     PORTA,W             ;Move RA0 into C
0027 0368           00399           RLF     I2CREG,F            ;Shift in bit
0028 0603           00400           SKPNC                       ;Skip if not done
0029 0A36           00401           GOTO    ACK_I2C             ;Acknowledge byte
                    00402
002A 0608           00403           BTFSC   I2CREG,0            ;Skip if data bit was 0
002B 0A31           00404           GOTO    ii_1                ;This bit was set
002C                00405 ii_0
                    00406           BBC     B_SCL,loop4         ;Wait for clock  low
002C 0725               M           BTFSS   B_SCL>>8,B_SCL&7    ;(macro)
002D 0A24               M           GOTO    loop4               ;(macro)
                    00407           SKBS    B_SDA               ;Data low-high == stop
002E 0705               M           BTFSS   B_SDA>>8,B_SDA&7    ;(macro)
002F 0A2C           00408           GOTO    ii_0
0030                00409 I2CSTOP
                    00410           USER_MSG                    ;process completed message!
                        M ;*** This would be user code to process a message
0030 0A11           00411           GOTO    I2CWAIT             ;back to main loop
                    00412
                    00413 ii_1      BBC     B_SDA,I2CWAIT       ;Data high-low == start
0031 0705               M           BTFSS   B_SDA>>8,B_SDA&7    ;(macro)
0032 0A11               M           GOTO    I2CWAIT             ;(macro)
```


© 1997 Microchip Technology Inc.

```
                00414           BBC     B_SCL,loop4         ;Wait for clock  low
0033 0725           M           BTFSS   B_SCL>>8,B_SCL&7   ;(macro)
0034 0A24           M           GOTO    loop4               ;(macro)
0035 0A31       00415           GOTO    ii_1
                00416
0036            00417 ACK_I2C
                00418           BBC     B_UA,ACK_UA         ;Not addressed - check unit address
0036 0727           M           BTFSS   B_UA>>8,B_UA&7      ;(macro)
0037 0A8B           M           GOTO    ACK_UA              ;(macro)
                00419           BBS     B_SA,ACK_SA         ;Reading secondary address
0038 0647           M           BTFSC   B_SA>>8,B_SA&7      ;(macro)
0039 0A97           M           GOTO    ACK_SA              ;(macro)
                00420
                00421 ;****
                00422 ;** Do what must be done with new data bytes here (before ACKloop)
                00423 ;** Don't ack if byte can't be processed!
                00424 ;****
                00425 ;--------
                00426
                00427           USER_RECV
                    M ;*** This would be user code to process a received byte
                    M ;*** example code sends sub-address 0 to port b
                    M           BBC     B_ID,_NXI_notid ;Channel 0! Bit set if INITIAL address was 0
003A 0767           M           BTFSS   B_ID>>8,B_ID&7      ;(macro)
003B 0A3F           M           GOTO    _NXI_notid          ;(macro)
003C 0208           M           MOVFW   I2CREG              ;get received byte
003D 0026           M           MOVWF   PORTB               ;and write it on portb
003E 0A47           M           GOTO    IN_CONT
003F                M _NXI_notid
                00428
003F 0C07       00429           MOVLW   07H                 ;Register count
0040 0169       00430           ANDWF   I2CSUBA,f           ;Limit register count
0041 0C0B       00431           MOVLW   I2CR0               ;Pseudo-registers
0042 01C9       00432           ADDWF   I2CSUBA,W           ;Offset from buffer start
0043 02A9       00433           INCF    I2CSUBA, F          ;Next sub-address
0044 0024       00434           MOVWF   FSR                 ;Indirect address
0045 0208       00435           MOVFW   I2CREG
0046 0020       00436           MOVWF   ind                 ;Put data into register
                00437
0047            00438 IN_CONT                               ;continue point for intercepted bytes
                00439
0047            00440 ACKloop
                00441           BBS     B_SCL,ACKloop       ;Wait for clock low
0047 0625           M           BTFSC   B_SCL>>8,B_SCL&7    ;(macro)
0048 0A47           M           GOTO    ACKloop             ;(macro)
                00442
                00443           CLB     B_SDA               ;Set ACK
0049 0405           M           BCF     B_SDA>>8,B_SDA&7    ;(macro)
004A 0CF6       00444           MOVLW   TAWRITE
004B 0005       00445           TRIS    PORTA
                00446           CLB     B_SDA               ;Set ACK (just in case docs are wrong)
004C 0405           M           BCF     B_SDA>>8,B_SDA &7   ;(macro)
                00447
                00448
004D            00449 ACKloop2
                00450           USER_Q
                    M ;*** This would be quick user code
                00451           BBC     B_SCL,ACKloop2      ;Wait for clock high
004D 0725           M           BTFSS   B_SCL>>8,B_SCL&7    ;(macro)
004E 0A4D           M           GOTO    ACKloop2            ;(macro)
                00452
004F            00453 ACKloop3
                00454           USER_Q
                    M ;*** This would be quick user code
                00455           BBS     B_SCL,ACKloop3      ;Wait for clock low
004F 0625           M           BTFSC   B_SCL>>8,B_SCL&7    ;(macro)
```

```
0050 0A4F          M         GOTO    ACKloop3          ;(macro)
                   00456
0051 0CF7          00457     MOVLW   TAREAD            ;End ACK
0052 0005          00458     TRIS    PORTA
                   00459
                   00460     BBC     B_RD,NEXTBYTE   ;Skip if read (we were  acking address only)
0053 0707          M         BTFSS   B_RD>>8,B_RD&7          ;(macro)
0054 0A21          M         GOTO    NEXTBYTE                ;(macro)
                   00461
                   00462
                   00463 ;*****************************************************************
                   00464 ; I2C Readback (I2C read request)
                   00465 ; Application specific code to get bytes to send may be added here.
                   00466 ; This routine gets data from location pointed to by I2CSUBA and
                   00467 ; sends it to I2C. Subsequent reads get sequential addresses. This
                   00468 ; version AND's the register # with 7 to limit to 8 registers (for
                   00469 ; speed). Thiscould be modified to do a comparison to an Absolute
                   00470 ; number.
                   00471 ;*****************************************************************
                   00472
0055               00473 NEXTOUT
                   00474
                   00475 ;*** <<< PUT NEXT BYTE INTO I2CREG HERE NOW! >>> ***
                   00476
                   00477         USER_XMIT
                       M ;*** This would be user code to prepare an output byte
                       M ;*** example code sends id string to output
                       M         BBC     B_ID,_NXO_notid ;Channel 0! Bit set if INITIAL address was 0
0055 0767              M         BTFSS   B_ID>>8,B_ID&7  ;(macro)
0056 0A59              M         GOTO    _NXO_notid      ;(macro)
0057 0900              M         CALL    GETID           ;get next byte from ID channel
0058 0A60              M         GOTO    OUT_CONT        ;and send it
0059                   M _NXO_notid
                   00478
0059 0C07          00479         MOVLW   07H             ;Register count
005A 0169          00480         ANDWF   I2CSUBA,f       ;Limit register count
005B 0C0B          00481         MOVLW   I2CR0           ;Pseudo-registers
005C 01C9          00482         ADDWF   I2CSUBA,W       ;Offset from buffer start
005D 02A9          00483         INCF    I2CSUBA, F      ;Next sub-address
005E 0024          00484         MOVWF   FSR             ;Indirect address
005F 0200          00485         MOVFW   ind             ;Get data from register
                   00486
0060               00487 OUT_CONT
0060 0028          00488         MOVWF   I2CREG
                   00489 ;-- add code here to init I2CREG! when B_ID is clear!
0061 0C08          00490         MOVLW   8               ;Bit counter
0062 002A          00491         MOVWF   I2CBITS
                   00492
                   00493 ;** OUT bits! -- external poll may be executed during low clock cycle,
                   00494 ;   but may also be executed during high cycle if necessary.
                   00495
                   00496 ;* ENABLE line is checked for loss of enable ONLY during HIGH CLOCK
                   00497
                   00498 ;*** CLOCK IS LOW -- CHANGE DATA HERE FIRST!
                   00499
                   00500 ;*** loop 1: data was 1
0063               00501 iiOUT_loop_1
0063 0368          00502         RLF     I2CREG,F        ;Shift data out, MSB first
0064 0603          00503         SKPNC                   ;1->0: change
0065 0A79          00504         GOTO    iiOUT_1         ;Output another 1!
                   00505         CLB     B_SDA           ;Output 0
0066 0405              M         BCF     B_SDA>>8,B_SDA&7 ;(macro)
0067 0CF6          00506         MOVLW   TAWRITE
0068 0005          00507         TRIS    PORTA
                   00508         CLB     B_SDA           ;Set data (just in case docs are wrong)
0069 0405              M         BCF     B_SDA>>8,B_SDA&7 ;(macro)
```


© 1997 Microchip Technology Inc.

```
                00509
006A            00510 iiOUT_0
006A 0004       00511         CLRWDT                    ;Clear watchdog timer
                00512
                00513         USER_Q
                    M ;*** This would be quick user code
                00514
006B            00515 iiOUT_loop_02
                00516         BBC     B_SCL,iiOUT_loop_02 ;Wait for clock high
006B 0725           M         BTFSS   B_SCL>>8,B_SCL&7 ;(macro)
006C 0A6B           M         GOTO    iiOUT_loop_02    ;(macro)
                00517
                00518         USER_Q
                    M ;*** This would be quick user code
                00519
006D            00520 iiOUT_loop_03
                00521         BBS     B_SCL,iiOUT_loop_03 ;Wait for clock low
006D 0625           M         BTFSC   B_SCL>>8,B_SCL&7    ;(macro)
006E 0A6D           M         GOTO    iiOUT_loop_03       ;(macro)
                00522
006F 02EA       00523         DECFSZ  I2CBITS, F      ;Count bits
0070 0A74       00524         GOTO    iiOUT_loop_0    ;Loop for last bit 0
0071 0CF7       00525         MOVLW   TAREAD          ;Done with last bit 0... Set to 1 for ACK
0072 0005       00526         TRIS    PORTA
0073 0A80       00527         GOTO    iiOUT_ack       ;Get ACK
                00528
0074            00529 iiOUT_loop_0
0074 0368       00530         RLF     I2CREG,F        ;Shift data out, MSB first
0075 0703       00531         SKPC                    ;0->1: change
0076 0A6A       00532         GOTO    iiOUT_0         ;Output another 0!
                00533
0077 0CF7       00534         MOVLW   TAREAD          ;Set to 1
0078 0005       00535         TRIS    PORTA
                00536
0079            00537 iiOUT_1
0079 0004       00538         CLRWDT                  ;Clear watchdog timer
                00539
                00540
                00541         USER_Q
                    M ;*** This would be quick user code
                00542
007A            00543 iiOUT_loop_12
                00544         BBC     B_SCL,iiOUT_loop_12 ;Wait for clock high
007A 0725           M         BTFSS   B_SCL>>8,B_SCL&7    ;(macro)
007B 0A7A           M         GOTO    iiOUT_loop_12       ;(macro)
                00545
                00546
                00547         USER_Q
                    M ;*** This would be quick user code
                00548
007C            00549 iiOUT_loop_13
                00550         BBS     B_SCL,iiOUT_loop_13 ;Wait for clock low
007C 0625           M         BTFSC   B_SCL>>8,B_SCL&7    ;(macro)
007D 0A7C           M         GOTO    iiOUT_loop_13       ;(macro)
                00551
007E 02EA       00552         DECFSZ  I2CBITS, F          ;Count bits
007F 0A63       00553         GOTO    iiOUT_loop_1        ;Loop for last bit 1
                00554
                00555
0080            00556 iiOUT_ack ;Get acknowledge
0080 02A9       00557         INCF    I2CSUBA, F          ;Next sub-address
                00558
0081            00559 iiOUT_loop_a2
                00560         BBC     B_SCL,iiOUT_loop_a2 ;Wait for clock high
0081 0725           M         BTFSS   B_SCL>>8,B_SCL&7    ;(macro)
0082 0A81           M         GOTO    iiOUT_loop_a2       ;(macro)
```


© 1997 Microchip Technology Inc.

```
                00561
                00562           BBS     B_SDA,I2CWAIT   ;No ACK --- wait for restart!
0083 0605           M           BTFSC   B_SDA>>8,B_SDA&7     ;(macro)
0084 0A11           M           GOTO    I2CWAIT             ;(macro)
                00563
                00564 ;-- prepare next character here!
                00565
0085            00566 iiOUT_loop_a3
                00567           BBC     B_SCL,NEXTOUT   ;Wait for clock low - output next char!
0085 0725           M           BTFSS   B_SCL>>8,B_SCL&7     ;(macro)
0086 0A55           M           GOTO    NEXTOUT             ;(macro)
                00568
                00569           BBS     B_SDA,iiOUT_loop_a3 ;Watch out for new start condition!
0087 0605           M           BTFSC   B_SDA>>8,B_SDA&7     ;(macro)
0088 0A85           M           GOTO    iiOUT_loop_a3       ;(macro)
0089 0A11       00570           GOTO    I2CWAIT             ;Stop received!
008A            00571 USER_READ                             ;user code to process data sent
                00572
008A 0A11       00573           GOTO    I2CWAIT
                00574
                00575
                00576 ;*****************************************************************
                00577 ;* Unit address received - check for valid address
                00578 ;*
                00579 ;*****************************************************************
008B            00580 ACK_UA
                00581           SEB     B_UA                ;Flag unit address received
008B 0527           M           BSF     B_UA>>8,B_UA&7      ;(macro)
008C 0608       00582           BTFSC   I2CREG,0            ;Skip if data coming in
                00583           SEB     B_RD                ;Flag - reading from slave
008D 0507           M           BSF     B_RD>>8,B_RD&7      ;(macro)
008E 0208       00584           MOVF    I2CREG,W            ;Get address
008F 0EFE       00585           ANDLW   0FEH                ;Mask direction flage before compare
0090 0FD6       00586           XORLW   DEVICE_ADDRESS      ;Device address
0091 0743 0A11  00587           BNZ     I2CWAIT             ;Not for me! (skip rest of message)
                00588           BBS     B_RD,ACKloop        ;Read - no secondary address
0093 0607           M           BTFSC   B_RD>>8,B_RD&7      ;(macro)
0094 0A47           M           GOTO    ACKloop             ;(macro)
                00589           SEB     B_SA                ;Next is secondary address
0095 0547           M           BSF     B_SA>>8,B_SA&7      ;(macro)
0096 0A47       00590           GOTO    ACKloop             ;Yes! ACK address and continue!
                00591
                00592 ;*****************************************************************
                00593 ;* Secondary address received - stow it!
                00594 ;* SA = 0 is converted to 128 to facilitate ID read
                00595 ;*****************************************************************
0097            00596 ACK_SA
                00597           CLB     B_SA                ;Flag second address received
0097 0447           M           BCF     B_SA>>8,B_SA&7      ;(macro)
                00598           CLB     B_ID
0098 0467           M           BCF     B_ID>>8,B_ID&7      ;(macro)
0099 0208       00599           MOVFW   I2CREG              ;Get subaddress
009A 0643       00600           SKPNZ                       ;Not 0
                00601           SEB     B_ID                ;Flag - id area selected
009B 0567           M           BSF     B_ID>>8,B_ID&7      ;(macro)
009C 0029       00602           MOVWF   I2CSUBA             ;Set subaddress
009D 0A47       00603           GOTO    ACKloop
                00604
                00605
                00606           END
```


© 1997 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX-- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   159
Program Memory Words Free:   353


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Using PIC16C5X Microcontrollers as LCD Drivers

Author: *Al Lovrich*
Microchip Technology Inc.

INTRODUCTION

This application note describes an LCD controller implementation using a PIC16C55 microcontroller. This technique offers display capabilities for applications that require a small display at a low cost, together with the capabilities of the standard PIC16C55 microcontroller. We start by an overview of LCD devices and their theory of operation followed by software implementation issues of the controller. The source code for controlling a multiplexed LCD display is included in Appendix A.

LIQUID CRYSTAL DISPLAYS

The Liquid Crystal Display (LCD) is a thin layer of "Liquid Crystal Material" deposited between two plates of glass. The raw LCD is often referred to as "glass". Electrodes are attached to both sides of the glass. One side is referred to as common or backplane, while the other side is referred to as segment.

An LCD is modeled as a capacitor, with one side connected to the common plane and the other side connected to the segment, as shown in Figure 1. LCDs are sensitive to Root Mean Square Voltage (VRMS) levels. When a VRMS level of zero volts is applied to the LCD, the LCD is practically transparent.

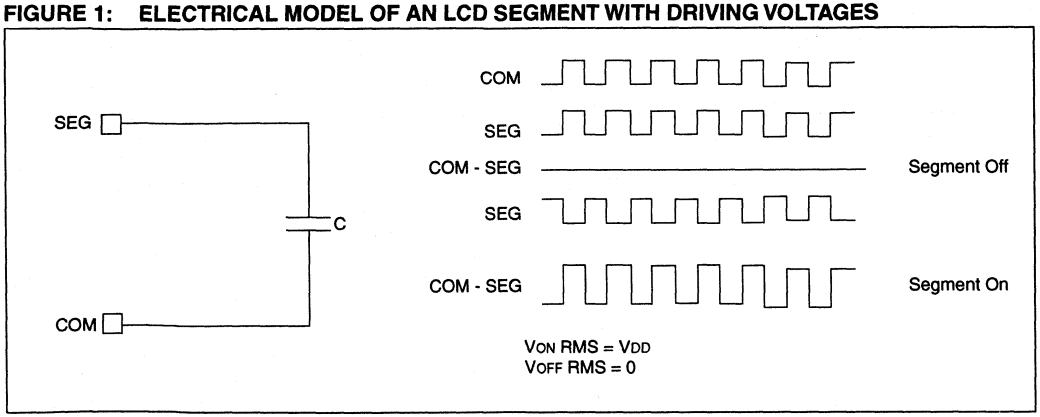
To turn an LCD segment "on," which makes the segment turn dark or opaque, an LCD RMS voltage that is greater than the LCD threshold voltage is applied to the LCD. The RMS LCD voltage is the RMS voltage across the capacitor *C* in Figure 1, which is equal to the potential difference between SEG and COM values.

Different LCDs have different characteristics; Figure 2 shows typical voltage vs relative contrast characteristics. Notations on the curve show operating points for multiplex operation with the threshold voltage set to 1.7 VRMS. This voltage is often used as the measure of voltage for the LCD to be "off" or transparent. The curve is normalized and assumes a viewing angle of 90° to the plane of the LCD.

Contrast control, the process of turning on a segment, is achieved by moving the operating point of the LCD. This is achieved by applying a voltage to the LCD that is greater than the LCDs threshold voltage. A typical circuit to accomplish this task is shown in Figure 3.

Driving a liquid crystal display at direct current (DC) will cause permanent damage to the display unit. In order to prevent irreversible electrochemical action from destroying the display, the voltage at all segment locations must reverse polarity periodically so that a zero net voltage is applied to the device. This process is referred to as AC voltage application. There are two LCD driving methods available: Static and multiplexed.

FIGURE 1: ELECTRICAL MODEL OF AN LCD SEGMENT WITH DRIVING VOLTAGES



© 1997 Microchip Technology Inc.

Conventional LCDs have separate external connections for each and every segment plus a common plane. This is the most basic method that results in good display quality. The main disadvantage of this driving method is that each segment requires one liquid crystal driver. The static driving method uses the frame frequency, defined as a period of the common plane signal, of several tens to several hundred Hz. A lower frequency would result in blinking effects and higher frequencies would increase power requirements. To turn a segment on, a voltage that has an opposite polarity to the common plane signal must be applied resulting in a large RMS voltage across the plates. To turn off a segment, a voltage that is of the same polarity to the common plane signal is applied. This drive method is universal to driving LCD segments. Figure 1 shows an example of this driving method.

The LCD frequency is defined as the rate of output changes of the common plane and segment signals, whereas the frame rate is defined as

$$F_{FRAME} = \frac{F_{FRAME}}{N}$$

where *N* is the multiplex rate or number of backplanes. Typically, FFRAME ranges from 25 Hz to 300 Hz. The most commonly used frame frequency is 40-70 Hz. A lower frequency would result in flicker effects and higher frequency would increase power requirements.

Multiplexed LCDs maintain their liquid crystal characteristics. They have a low power consumption, a high contrast ratio under high ambient light levels, and they reduce the number of external connections necessary for dot matrix and alphanumeric displays. The multiplex driving method reduces the number of driver circuits, or microcontroller I/O pins if a software method is used. The method of drive for multiplexed displays is Time Division Multiplex (TDM) with the number of time divisions equal to twice the number of common planes used in a given format. In order to prevent permanent damage to the LCD display, the voltage at all segment locations must reverse polarity periodically so that zero net voltage is applied. This is the reason for the doubling in time divisions; each common plane must be alternately driven with a voltage pulse of opposite polarity. The drive frequency should be greater than the flicker rate of 25 Hz. Since increasing the drive frequency significantly above this value increases current demand by the CMOS circuitry, an upper drive frequency level of 60 Hz is recommended by most LCD manufacturers. We have chosen a drive rate of 50 Hz for this application note which results in a frame period of 20 ms. The most commonly available formats are 2x4, 3x3, and 5x7. In this report we use a 2x4 format LCD to display hexadecimal digits.

FIGURE 2: TYPICAL LCD CHARATERISTICS

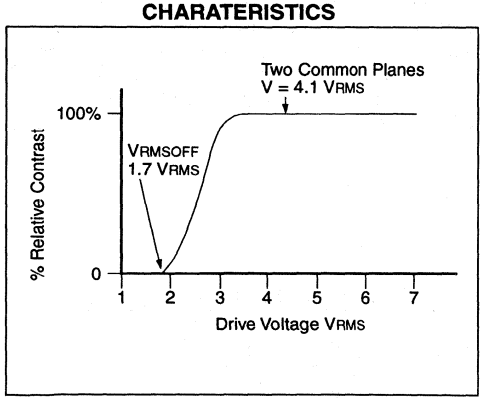
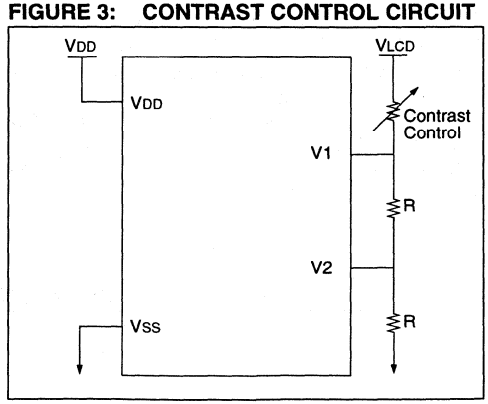


FIGURE 3: CONTRAST CONTROL CIRCUIT



© 1997 Microchip Technology Inc.

To better understand multiplexed LCD control it is best to look at the general case. The segments in a multiplexed LCD are arranged in an X-Y grid form as shown in Figure 4. The common plane signals maintain their relative shape at all times, as shown in Figure 5. To turn on segment 1 (SEG1), we need to apply a voltage V_D, such that $V_S + V_D$ turns the segment on and $V_S - V_D$ turns the segment off. Note that the segment signal V_D is symmetrical. This is a consequence of the intervals that the common plane signal is not present at all times. Use of nonsymmetrical waveform will result in a higher V_{RMS} present on the unaddressed segments.

The symmetrical nature of the waveforms theoretically result in zero DC voltage levels. CMOS drivers (e.g., microcontrollers) operate at 0 to +5V levels (rail voltage levels). This would require driving voltages beyond the range of operation. This constraint is addressed by a technique referred to as "level shifting" or "biasing". Level shifting allows application of voltages in the range of 0 to +2.5V, which is compatible with these drivers. This would require an additional voltage level of +2.5V, which can be implemented through a simple resistive voltage divider circuit.

FIGURE 4: MULTIPLEXED LCD SEGMENT ARRANGEMENT

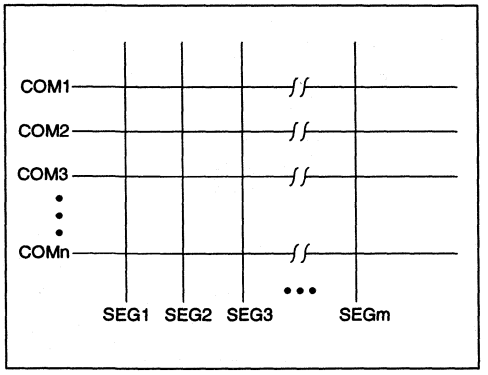
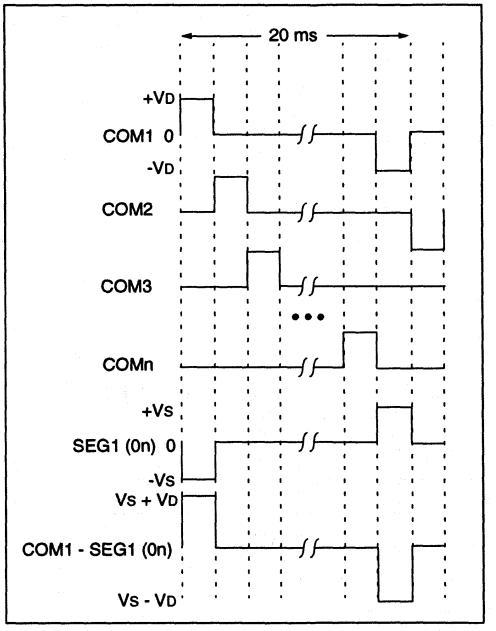


FIGURE 5: MULTIPLEXED LCD DRIVE WAVEFORMS



© 1997 Microchip Technology Inc.

IMPLEMENTATION

The ideas presented in the previous section can be applied to any size multiplexed LCD display. In our implementation we used a 4-digit LCD from Ocular Inc. [1]. The circuit diagram used in this application note is shown in Figure 6. Each I/O pin on the PIC16C55 device controls the state of two segments (Figure 6) which requires a total of 16 I/O pins. The reference voltages are generated through a simple resistive voltage divider circuit.

The voltage levels are generated by taking advantage of PIC16C5X I/O pin set to input, which tri-states the voltage level seen on the pin. This method uses 4 I/O pins to generate the proper voltage levels. Figure 7 shows the truth table for generating the voltage levels. Figure 8 shows how to create a bitmap for different digits. Figure 9 shows the waveforms generated for the accompanying software which implements a hexadecimal counter.

FIGURE 6: SYSTEM CONFIGURATION WITH LCD PINOUT

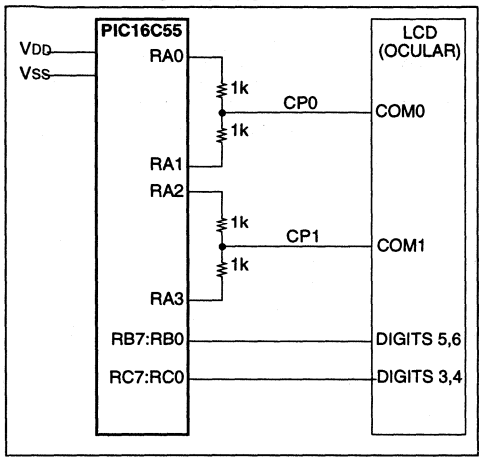


FIGURE 7: COMMON PLANE SIGNAL GENERATION

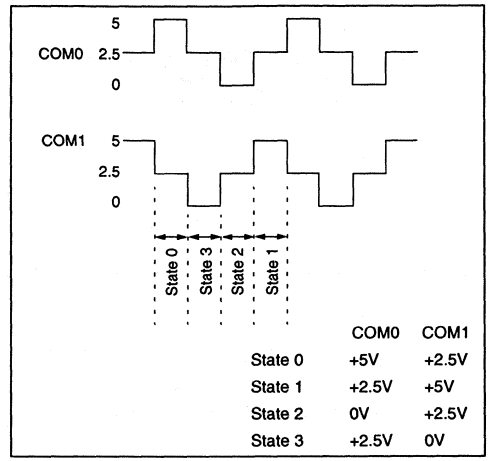
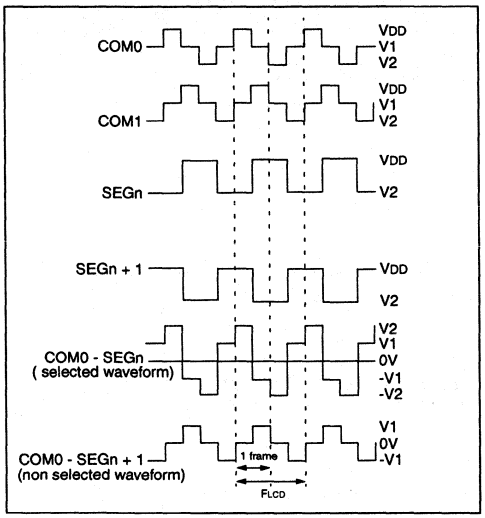


FIGURE 8: LCD CHARACTER BITMAP

Digit	COM0 SEG0				COM1 SEG1				COM0 SEG2				COM1 SEG3			
	F	E	D	DP	A	G	C	B	F	E	D	DP	A	G	C	B
0	0	0	0	1	0	0	1	0	1	1	1	0	1	1	0	1
1	1	1	1	1	1	0	1	0	0	0	0	0	0	1	0	1
2	1	0	0	1	0	0	0	1	0	1	1	0	1	1	1	0
3	1	1	0	1	0	0	0	0	0	0	1	0	1	1	1	1
4	0	1	1	1	1	0	0	0	1	0	0	0	0	1	1	1
5	0	1	0	1	0	1	0	0	1	0	1	0	1	0	1	1
6	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1
7	1	1	1	1	0	0	1	0	0	0	0	0	1	1	0	1
8	0	0	0	1	0	0	0	0	1	1	1	0	1	1	1	1
9	0	1	0	1	0	0	0	0	1	0	1	0	1	1	1	1
a	0	0	1	1	0	0	0	0	1	1	0	0	1	1	1	1
b	0	0	0	1	1	1	0	0	1	1	1	0	0	0	1	1
c	1	0	0	1	1	1	0	1	0	1	1	0	0	0	1	0
d	1	0	0	1	1	0	0	0	0	1	1	0	0	1	1	1
e	0	0	0	1	0	1	0	1	1	1	1	0	1	0	1	0
f	0	0	1	1	0	1	0	1	1	1	0	0	1	0	1	0

© 1997 Microchip Technology Inc.

FIGURE 9: EXAMPLE OF OUTPUT WAVEFORMS FOR DIGIT 4



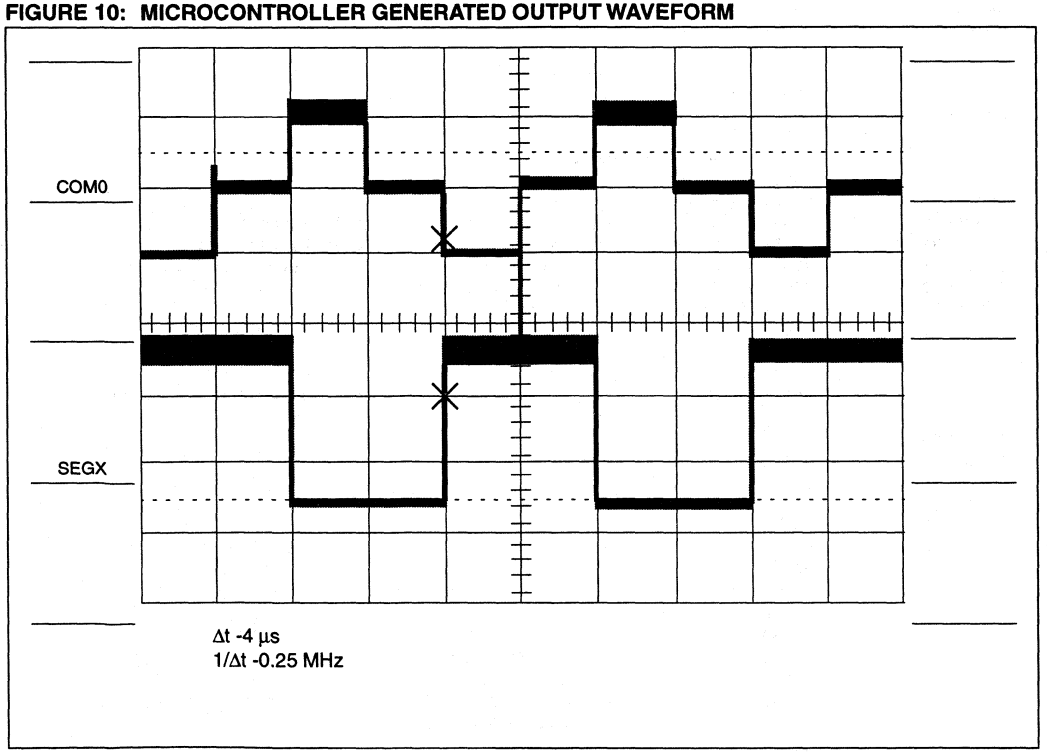
CONCLUSION

In this application note we have demonstrated the use of PIC16C5X devices to implement a simple LCD controller. As discussed earlier, it is important to keep the generated DC voltage to a minimum to extend the life of the LCD. Ideally one should switch all the I/O lines simultaneously, however, implementation of software of the LCD controller will necessarily introduce a delay which is proportional to the instruction cycle of the microcontroller, as shown in Figure 10. Therefore it is necessary to keep the switching time to a minimum. Our implementation introduced less than 50 mV of DC voltage on the segment lines which is below the manufacturer's recommended DC offset voltage of 60 mV.

REFERENCES

[1] Ocular Inc., Drawing number JH074.

FIGURE 10: MICROCONTROLLER GENERATED OUTPUT WAVEFORM



Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: PIC16C5X AS A MULTIPLEXED LCD DRIVER

```
MPASM 01.40 Released              LCD.ASM   1-16-1997  16:45:44         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001  LIST    C=132,n=0,p=16c55,r=dec
                     00002
                     00003 ;*****************************************************************
                     00004 ; Project: PIC16C5X as a multiplexed LCD driver.                  *
                     00005 ;                                                                *
                     00006 ; Revision history:                                              *
                     00007 ;       04/14/93        original                                 *
                     00008 ;       1-15-97         Compatability with MPASMWIN 1.40         *
                     00009 ;*****************************************************************
                     00010
                     00011 ; Equates
                     00012
  000001FF           00013 pic54        equ  0x1ff           ; Define Reset Vectors
  000001FF           00014 pic55        equ  0x1ff
  000003FF           00015 pic56        equ  0x3ff
  000007FF           00016 pic57        equ  0x7ff
                     00017
  00000001           00018 TMR0         equ  1               ; f1
  00000002           00019 pc           equ  2               ; f2
  00000003           00020 status       equ  3               ; f3
  00000004           00021 fsr          equ  4               ; f4
                     00022
  00000005           00023 porta        equ  5               ; f5
  00000006           00024 portb        equ  6               ; f6
  00000007           00025 portc        equ  7               ; f8
                     00026
                     00027 ; realtime mode registers
                     00028
  00000008           00029 currentState equ  8
  00000009           00030 msTimer      equ  currentState+1  ; Millisecond timer
  0000000A           00031 sTimerLow    equ  msTimer+1       ; Lower byte second timer
  0000000B           00032 sTimerHigh   equ  sTimerLow+1     ; Upper byte second timer
  0000000C           00033 digit56      equ  sTimerHigh+1
  0000000D           00034 digit34      equ  digit56+1
                     00035
                     00036 ; Misc definitions
                     00037
  00000060           00038 FIVEMSEC     equ  96              ; Assuming 4.096 MHz crystal
                     00039
  00000000           00040 w            equ  0
  00000001           00041 f            equ  1
                     00042
  00000002           00043 z            equ  2
                     00044
                     00045 ; Status register bits
                     00046
                     00047 ;*****************************************************
                     00048 ; Port assignments                                   *
                     00049 ;                                                    *
                     00050 ;  porta - bit0: Common Plane 0                      *
                     00051 ;           bit1: Common Plane 0                     *
                     00052 ;           bit2: Common Plane 1                     *
                     00053 ;           bit3: Common Plane 1                     *
                     00054 ;                                                    *
```


© 1997 Microchip Technology Inc.

```
                00055 ;  portb - bit0: 6B/DP                                  *
                00056 ;          bit1: 6C/6D                                  *
                00057 ;          bit2: 6G/6E                                  *
                00058 ;          bit3: 6A/6F                                  *
                00059 ;          bit4: 5B/DP                                  *
                00060 ;          bit5: 5C/5D                                  *
                00061 ;          bit6: 5G/5E                                  *
                00062 ;          bit7: 5A/5F                                  *
                00063 ;                                                       *
                00064 ;  portc - bit0: 4B/DP                                  *
                00065 ;        - bit1: 4C/4D                                  *
                00066 ;        - bit2: 4G/4E                                  *
                00067 ;        - bit3: 4A/4F                                  *
                00068 ;        - bit4: 3B/DP                                  *
                00069 ;        - bit5: 3C/3D                                  *
                00070 ;        - bit6: 3G/3E                                  *
                00071 ;        - bit7: 3A/3F                                  *
                00072 ;                                                       *
                00073 ;***************************************************
                00074
                00075 ;***************************************************
                00076 ;               Macro definitions                       *
                00077 ;***************************************************
                00078 UpdateState     macro   State, Table
                00079
                00080         swapf   sTimerLow, w
                00081         andlw   0xf               ; Isolate digit 5 (offset)
                00082         call    Table
                00083         movwf   digit56
                00084         swapf   digit56, f
                00085
                00086         movf    sTimerLow, w
                00087         andlw   0xf               ; Isolate digit 6 (offset)
                00088         call    Table
                00089         iorwf   digit56, f
                00090
                00091         swapf   sTimerHigh, w
                00092         andlw   0xf               ; Isolate digit 5 (offset)
                00093         call    Table
                00094         movwf   digit34
                00095         swapf   digit34, f
                00096
                00097         movf    sTimerHigh, w
                00098         andlw   0xf               ; Isolate digit 6 (offset)
                00099         call    Table
                00100         iorwf   digit34, f
                00101
                00102         movf    digit34, w        ; Display digits 3 & 4
                00103         movwf   portc
                00104         movf    digit56, w
                00105         movwf   portb             ; Display digits 5 & 6
                00106         endm
                00107
0000            00108         org     0
                00109
                00110 ; Initialize ports A, B, and C and TMR0. In case of output data
                00111 ; values, set the data latch first, then set the port direction.
                00112
0000            00113 Initialize
0000 0C01       00114         movlw   00000001b         ; Set data latch
0001 0025       00115         movwf   porta
0002 0C08       00116         movlw   00001000b         ; Set I/O direction
0003 0005       00117         tris    porta
                00118
0004 0C00       00119         movlw   00000000b         ; Set levels to low
0005 0026       00120         movwf   portb
```

```
0006 0C00          00121         movlw   00000000b         ; Set as outputs
0007 0006          00122         tris    portb
                   00123
0008 0C00          00124         movlw   00000000b         ; Set levels to low
0009 0027          00125         movwf   portc
000A 0C00          00126         movlw   00000000b         ; Set as outputs
000B 0007          00127         tris    portc
                   00128
000C 0C04          00129         movlw   0x04              ; Set prescaler
000D 0002          00130         option
                   00131
000E 0C60          00132         movlw   FIVEMSEC          ; TMR0 = 5ms
000F 0021          00133         movwf   TMR0
                   00134
0010 0C04          00135         movlw   4
0011 0028          00136         movwf   currentState
                   00137
0012 0C0D          00138         movlw   0xd
0013 0029          00139         movwf   msTimer           ; Initialize millisecond timer
                   00140
0014 006A          00141         clrf    sTimerLow         ; Clear second counter
0015 006B          00142         clrf    sTimerHigh
                   00143
0016 0800          00144         retlw   0
                   00145
                   00146 ; Check timer register for timing out (TMR0 = 0). Remain in the
                   00147 ; loop until the timer times out.
                   00148
                   00149
                   00150 ; Wait for 5ms timer timeout
                   00151
0017               00152 Timer_Check
0017 0201          00153         movf    TMR0, w
0018 0743          00154         btfss   status, z
0019 0A17          00155         goto    Timer_Check
                   00156
001A 0C60          00157         movlw   FIVEMSEC
001B 0021          00158         movwf   TMR0
                   00159
001C 02E9          00160         decfsz  msTimer, f
001D 0A21          00161         goto    Update_Backplane
                   00162
001E 03EA          00163         incfsz  sTimerLow, f      ; Update second counter
001F 0A21          00164         goto    Update_Backplane
0020 02AB          00165         incf    sTimerHigh, f
                   00166
                   00167 ; RA0 and RA1 are used to control voltage level for common plane 0.
                   00168 ; RA2 and RA3 are used to control voltage level for common plane 1.
                   00169 ; There are four possible states with different voltage levels as
                   00170 ; follows:
                   00171 ;
                   00172 ; State 0 - cp0 = +5v           ra0=1, ra1=x
                   00173 ;           cp1 = +2.5v         ra2=1, ra3=0
                   00174 ; State 1 - cp0 = +2.5v         ra0=1, ra1=0
                   00175 ;           cp1 = +5v           ra2=1, ra3=x
                   00176 ; State 2 - cp0 =  0v           ra0=0, ra1=x
                   00177 ;           cp1 = +2.5v         ra2=1, ra3=0
                   00178 ; State 3 - cp0 = +2.5v         ra0=1, ra1=0
                   00179 ;           cp1 =  0v           ra2=0, ra3=x
                   00180
                   00181
0021               00182 Update_Backplane
0021 0004          00183         clrwdt                    ; Reset watchdog timer
                   00184
0022 00C8          00185         decf    currentState, w   ; Update w register
0023 0E03          00186         andlw   0x03              ; Use only bit0/1
```


© 1997 Microchip Technology Inc.

```
0024 0028           00187         movwf   currentState      ; Update currentState
0025 01E2           00188         addwf   pc, f
                    00189
0026 0AAD           00190         goto    State3
0027 0A81           00191         goto    State2
0028 0A55           00192         goto    State1
                    00193 ;       goto    State0
                    00194
                    00195 ; State 0
                    00196
0029                00197 State0
                    00198         UpdateState     State0, S0_Table
                        M
0029 038A               M         swapf   sTimerLow, w
002A 0E0F               M         andlw   0xf               ; Isolate digit 5 (offset)
002B 0944               M         call    S0_Table
002C 002C               M         movwf   digit56
002D 03AC               M         swapf   digit56, f
                        M
002E 020A               M         movf    sTimerLow, w
002F 0E0F               M         andlw   0xf               ; Isolate digit 6 (offset)
0030 0944               M         call    S0_Table
0031 012C               M         iorwf   digit56, f
                        M
0032 038B               M         swapf   sTimerHigh, w
0033 0E0F               M         andlw   0xf               ; Isolate digit 5 (offset)
0034 0944               M         call    S0_Table
0035 002D               M         movwf   digit34
0036 03AD               M         swapf   digit34, f
                        M
0037 020B               M         movf    sTimerHigh, w
0038 0E0F               M         andlw   0xf               ; Isolate digit 6 (offset)
0039 0944               M         call    S0_Table
003A 012D               M         iorwf   digit34, f
                        M
003B 020D               M         movf    digit34, w        ; Display digits 3 & 4
003C 0027               M         movwf   portc
003D 020C               M         movf    digit56, w
003E 0026               M         movwf   portb             ; Display digits 5 & 6
                    00199
003F 0C05           00200         movlw   00000101b
0040 0025           00201         movwf   porta
0041 0C02           00202         movlw   00000010b
0042 0005           00203         tris    porta
                    00204
0043 0800           00205         retlw   0
                    00206
0044                00207 S0_Table
0044 01E2           00208         addwf   pc, f             ; Add offset to pc
                    00209
0045 0804           00210         retlw   0100b             ; 0
0046 080C           00211         retlw   1100b             ; 1
0047 0802           00212         retlw   0010b             ; 2
0048 0800           00213         retlw   0000b             ; 3
0049 0808           00214         retlw   1000b             ; 4
004A 0801           00215         retlw   0001b             ; 5
004B 080F           00216         retlw   1111b             ; 6
004C 0804           00217         retlw   0100b             ; 7
004D 0800           00218         retlw   0000b             ; 8
004E 0800           00219         retlw   0000b             ; 9
004F 0800           00220         retlw   0000b             ; a
0050 0809           00221         retlw   1001b             ; b
0051 080B           00222         retlw   1011b             ; c
0052 0808           00223         retlw   1000b             ; d
0053 0803           00224         retlw   0011b             ; e
0054 0803           00225         retlw   0011b             ; f
```


© 1997 Microchip Technology Inc.

```
                    00226
                    00227 ; State 1
                    00228
0055                00229 State1
                    00230         UpdateState     State1, S1_Table
                        M
0055 038A               M         swapf   sTimerLow, w
0056 0E0F               M         andlw   0xf               ; Isolate digit 5 (offset)
0057 0970               M         call    S1_Table
0058 002C               M         movwf   digit56
0059 03AC               M         swapf   digit56, f
                        M
005A 020A               M         movf    sTimerLow, w
005B 0E0F               M         andlw   0xf               ; Isolate digit 6 (offset)
005C 0970               M         call    S1_Table
005D 012C               M         iorwf   digit56, f
                        M
005E 038B               M         swapf   sTimerHigh, w
005F 0E0F               M         andlw   0xf               ; Isolate digit 5 (offset)
0060 0970               M         call    S1_Table
0061 002D               M         movwf   digit34
0062 03AD               M         swapf   digit34, f
                        M
0063 020B               M         movf    sTimerHigh, w
0064 0E0F               M         andlw   0xf               ; Isolate digit 6 (offset)
0065 0970               M         call    S1_Table
0066 012D               M         iorwf   digit34, f
                        M
0067 020D               M         movf    digit34, w        ; Display digits 3 & 4
0068 0027               M         movwf   portc
0069 020C               M         movf    digit56, w
006A 0026               M         movwf   portb             ; Display digits 5 & 6
                    00231
                    00232
006B 0C05           00233         movlw   00000101b
006C 0025           00234         movwf   porta
006D 0C08           00235         movlw   00001000b
006E 0005           00236         tris    porta
                    00237
006F 0800           00238         retlw   0
                    00239
0070                00240 S1_Table
0070 01E2           00241         addwf   pc, f
                    00242
0071 0801           00243         retlw   0001b             ; 0
0072 080F           00244         retlw   1111b             ; 1
0073 0809           00245         retlw   1001b             ; 2
0074 080D           00246         retlw   1101b             ; 3
0075 0807           00247         retlw   0111b             ; 4
0076 0805           00248         retlw   0101b             ; 5
0077 080F           00249         retlw   1111b             ; 6
0078 080F           00250         retlw   1111b             ; 7
0079 0801           00251         retlw   0001b             ; 8
007A 0805           00252         retlw   0101b             ; 9
007B 0803           00253         retlw   0011b             ; a
007C 0801           00254         retlw   0001b             ; b
007D 0809           00255         retlw   1001b             ; c
007E 0809           00256         retlw   1001b             ; d
007F 0801           00257         retlw   0001b             ; e
0080 0803           00258         retlw   0011b             ; f
                    00259
                    00260 ; State 2
                    00261
0081                00262 State2
                    00263         UpdateState     State2, S2_Table
                        M
```


© 1997 Microchip Technology Inc.

```
0081 038A               M         swapf   sTimerLow, w
0082 0E0F               M         andlw   0xf               ; Isolate digit 5 (offset)
0083 099C               M         call    S2_Table
0084 002C               M         movwf   digit56
0085 03AC               M         swapf   digit56, f
                        M
0086 020A               M         movf    sTimerLow, w
0087 0E0F               M         andlw   0xf               ; Isolate digit 6 (offset)
0088 099C               M         call    S2_Table
0089 012C               M         iorwf   digit56, f
                        M
008A 038B               M         swapf   sTimerHigh, w
008B 0E0F               M         andlw   0xf               ; Isolate digit 5 (offset)
008C 099C               M         call    S2_Table
008D 002D               M         movwf   digit34
008E 03AD               M         swapf   digit34, f
                        M
008F 020B               M         movf    sTimerHigh, w
0090 0E0F               M         andlw   0xf               ; Isolate digit 6 (offset)
0091 099C               M         call    S2_Table
0092 012D               M         iorwf   digit34, f
                        M
0093 020D               M         movf    digit34, w        ; Display digits 3 & 4
0094 0027               M         movwf   portc
0095 020C               M         movf    digit56, w
0096 0026               M         movwf   portb             ; Display digits 5 & 6
                     00264
0097 0C04            00265         movlw   00000100b
0098 0025            00266         movwf   porta
0099 0C02            00267         movlw   00000010b
009A 0005            00268         tris    porta
                     00269
009B 0800            00270         retlw   0
                     00271
009C                 00272 S2_Table
009C 01E2            00273         addwf   pc, f
                     00274
009D 080B            00275         retlw   1011b             ; 0
009E 0803            00276         retlw   0011b             ; 1
009F 080D            00277         retlw   1101b             ; 2
00A0 080F            00278         retlw   1111b             ; 3
00A1 0807            00279         retlw   0111b             ; 4
00A2 080E            00280         retlw   1110b             ; 5
00A3 080E            00281         retlw   1110b             ; 6
00A4 080B            00282         retlw   1011b             ; 7
00A5 080F            00283         retlw   1111b             ; 8
00A6 080F            00284         retlw   1111b             ; 9
00A7 080F            00285         retlw   1111b             ; a
00A8 0806            00286         retlw   0110b             ; b
00A9 0804            00287         retlw   0100b             ; c
00AA 0807            00288         retlw   0111b             ; d
00AB 080C            00289         retlw   1100b             ; e
00AC 080C            00290         retlw   1100b             ; f
                     00291
                     00292 ; State 3
                     00293
00AD                 00294 State3
                     00295         UpdateState     State3, S3_Table
                        M
00AD 038A               M         swapf   sTimerLow, w
00AE 0E0F               M         andlw   0xf                   ; Isolate digit 5 (offset)
00AF 09C8               M         call    S3_Table
00B0 002C               M         movwf   digit56
00B1 03AC               M         swapf   digit56, f
                        M
00B2 020A               M         movf    sTimerLow, w
```

```
00B3 0E0F               M         andlw   0xf                   ; Isolate digit 6 (offset)
00B4 09C8               M         call    S3_Table
00B5 012C               M         iorwf   digit56, f
                        M
00B6 038B               M         swapf   sTimerHigh, w
00B7 0E0F               M         andlw   0xf                   ; Isolate digit 5 (offset)
00B8 09C8               M         call    S3_Table
00B9 002D               M         movwf   digit34
00BA 03AD               M         swapf   digit34, f
                        M
00BB 020B               M         movf    sTimerHigh, w
00BC 0E0F               M         andlw   0xf                   ; Isolate digit 6 (offset)
00BD 09C8               M         call    S3_Table
00BE 012D               M         iorwf   digit34, f
                        M
00BF 020D               M         movf    digit34, w            ; Display digits 3 & 4
00C0 0027               M         movwf   portc
00C1 020C               M         movf    digit56, w
00C2 0026               M         movwf   portb                 ; Display digits 5 & 6
                     00296
00C3 0C01            00297         movlw   00000001b
00C4 0025            00298         movwf   porta
00C5 0C08            00299         movlw   00001000b
00C6 0005            00300         tris    porta
                     00301
00C7 0800            00302         retlw   0
                     00303
00C8                 00304 S3_Table
00C8 01E2            00305         addwf   pc, f
                     00306
00C9 080E            00307         retlw   1110b                 ; 0
00CA 0800            00308         retlw   0000b                 ; 1
00CB 0806            00309         retlw   0110b                 ; 2
00CC 0802            00310         retlw   0010b                 ; 3
00CD 0808            00311         retlw   1000b                 ; 4
00CE 080A            00312         retlw   1010b                 ; 5
00CF 080E            00313         retlw   1110b                 ; 6
00D0 0800            00314         retlw   0000b                 ; 7
00D1 080E            00315         retlw   1110b                 ; 8
00D2 080A            00316         retlw   1010b                 ; 9
00D3 080C            00317         retlw   1100b                 ; a
00D4 080E            00318         retlw   1110b                 ; b
00D5 0806            00319         retlw   0110b                 ; c
00D6 0806            00320         retlw   0110b                 ; d
00D7 080E            00321         retlw   1110b                 ; e
00D8 080C            00322         retlw   1100b                 ; f
                     00323
                     00324 ; Main code
                     00325
00D9                 00326 Start
00D9 0900            00327         call    Initialize
00DA                 00328 Repeat
00DA 0917            00329         call    Timer_Check
00DB 0ADA            00330         goto    Repeat
                     00331
01FF                 00332         org     pic55
                     00333
01FF                 00334 System_Reset
01FF 0AD9            00335         goto    Start
                     00336
                     00337         END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
```


© 1997 Microchip Technology Inc.

```
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXX---- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   221
Program Memory Words Free:   291


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

PIC16C54A EMI Results

Author: *Mark Palmer*
Ohmmeter Board Setup: *Stan D'Souza*
Testing: *Compatible Electronics*

INTRODUCTION

This paper discusses the EMI results of the PIC16C54A. These measurements were taken by an independent consulting firm that specializes in electromagnetic testing. These results are for a specific system design, each design will have its own results.

DEVICES USED

These tests were done on a randomly selected PIC16C54A device, and should be considered as typical results. Initial testing was done on three boards/devices. The device frequency of the boards were, respectively, 32 kHz, 4 MHz, and 20 MHz. As would be expected there was a substantial difference in EMI lives (decrease) at the low frequency as compared to the higher frequencies. The difference between the 4 MHz operation and the 20 MHz operation was marginal. The final testing of the device was done at 4 MHz, and was according to the FCC measurement procedure MP-4.

SYSTEM USED

The PIC16C54A device was tested in the Microchip OHMMETER board. This board is a three layer board, with a ground plane. Power and ground planes greatly help in the compliance of designs to the FCC part 15 subpart B testing. This board had minimal external components, so that the electromagnetic measurements could mostly be attributed to the device and design of the system. To reduce the noise that comes from a power supply, a 10 nF bypass capacitor was attached to the power / ground of the input jack.

EMI TESTING

The testing of electromagnetic noise (EMI) on a system has two types of commonly used testing environments, internal and external. The internal test is performed in a screen room to reduce the amount of stray EMI. The indoor test is useful in determining the source of EMI radiation. The external test is performed outdoors. This places the equipment under test in an environment to measure radiated emissions (EMI). The FCC only requires the outdoor testing of devices. The Equipment Under Test (EUT) was positioned to maximize the emissions.

INTERNAL TESTS

The Equipment Under Test (EUT) was placed on a wooden test bench, inside a screen room, 0.8 meters above the earth ground plane (Figure 1). The EUT was powered through the Line Impedance Stabilization Network (LISN) bonded to the ground plane. The LISN power was filtered and the filter was bonded to the ground plane. The EUT was positioned on the table with the minimum distance from any conductive surface, as specified in MP-4. The excess power cord was wrapped in a figure-8 pattern to form a bundle approximately 8 cm in length. The EUT configuration was set for the highest emission frequency, and data was collected under the program control of the host computer. The spectrum analyzer collected the maximum peak readings over each spectrum. The six highest emission levels and corresponding frequencies were sorted and are listed in Table 1.

TABLE 1: CONDUCTED EMISSIONS RESULTS

Frequency (MHz)	Emission Level dBμV	Emission Level μV	Specification Limit μV
0.5345	33.2	46	250
7.085	33.1	45	250
10.08	33.1	45	250
10.42	33.3	46	250
11.62	33.4	47	250
13.41	33.8	49	250

© 1996 Microchip Technology Inc.

While in the screen room additional analyses on the device was done. First a local probe was used to test the emission levels around the board and device. There were no measurable emissions at the I/O pins or their corresponding trace lines. Emissions were measured at the jack to the power supply. The power supply and cord were the greatest source of emissions for the EUT. This is due to the power cord being an antenna which emitted the noise from the power supply. Designers should attempt to minimize antennas which emit EMI. Antennas could be the power supply cord, as well as traces on the system board.

Second the PIC16C54A was monitored for susceptibility, following the IEC 801-3 specification. This test was measured from 27 MHz to 500 MHz in 10 kHz steps. The device did not display any signs of susceptibility (Table 2).

EXTERNAL TESTS

The open field site used for radiated emission testing was set up according to FCC bulletin OST 55.

Figure 2 shows the layout of the open field test site. The EUT was mounted on a turntable. The position of the turntable was remotely controlled to determine the highest emission levels. Initial testing was done with a broad band mounted on the antenna mast at a distance of 3 meters. Further investigation was done to determine the EUT positions that produced the maximum level of emissions. The receiving antenna was mounted on the antenna mast. The antenna height was varied to find the highest level of radiated emissions at each frequency. The six highest emission levels and corresponding frequencies were sorted and are listed in Table 3. Figure A-1 and Figure A-2 show the dBμV vs. MHz graphs.

TABLE 2: RADIATED SUSCEPTIBILITY

Frequency (MHz)	SPEC V/M	Threshold V/M	Modulation	Comments
27.0 - 500.0(1)	3.0	> 3.0	80%	No Susceptibility

† Frequency incremented in 10 kHz steps

TABLE 3: RADIATED EMISSIONS RESULTS

Frequency	Meter Reading (dBμV)	Antenna Factor dB	Effective Gain dB	Distribution Factor dB	Corr. Rdg. dBμV/m	Corr. Rdg. μV/m	Spec Limit μV/m
36.01	53.1	11.8	34.0	0	30.9	35	100
40.04	53.0	11.3	34.2	0	30.1	32	100
44.04	56.1	11.1	34.0	0	33.2	46	100
48.04	55.1	11.0	33.7	0	32.4	42	100
64.05	55.2	9.0	33.9	0	30.3	33	100
112.05	56.0	10.6	33.4	0	33.2	46	150

© 1996 Microchip Technology Inc.

FIGURE 1:

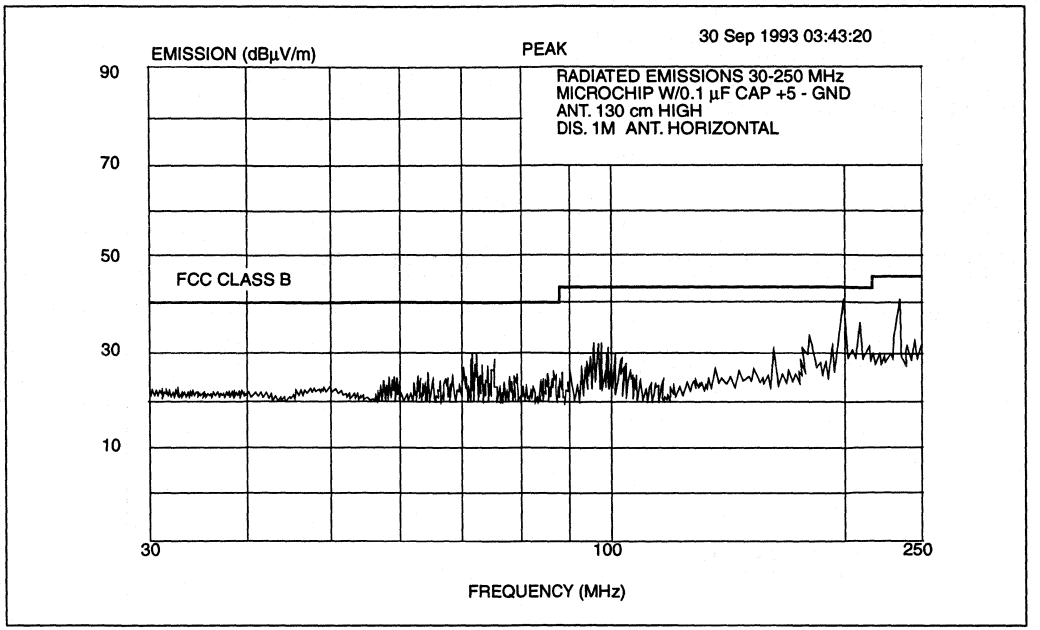
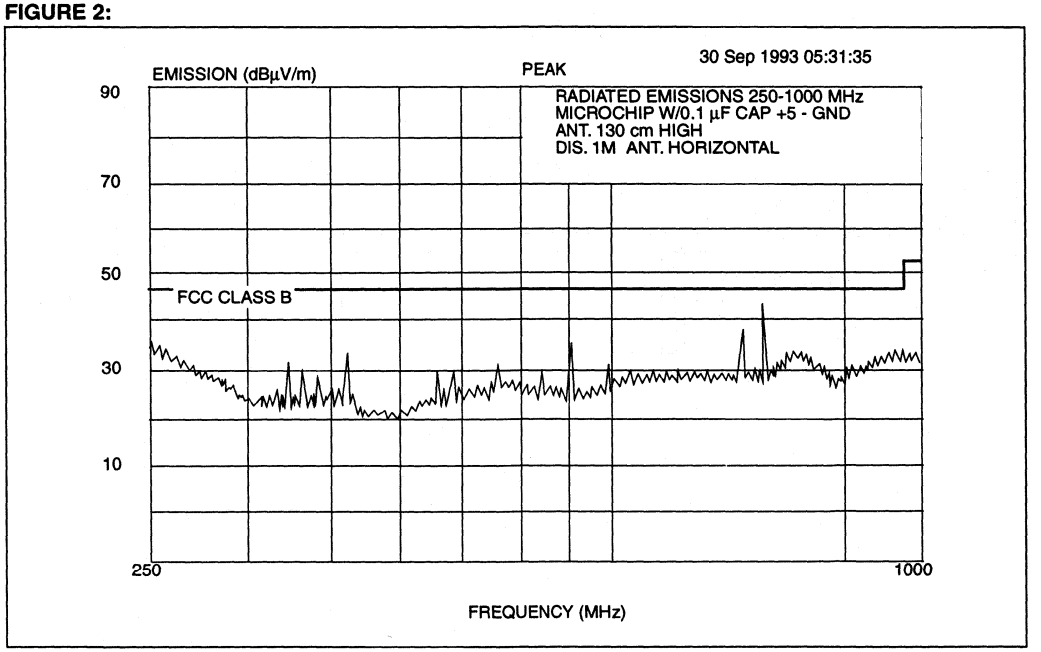


FIGURE 2:



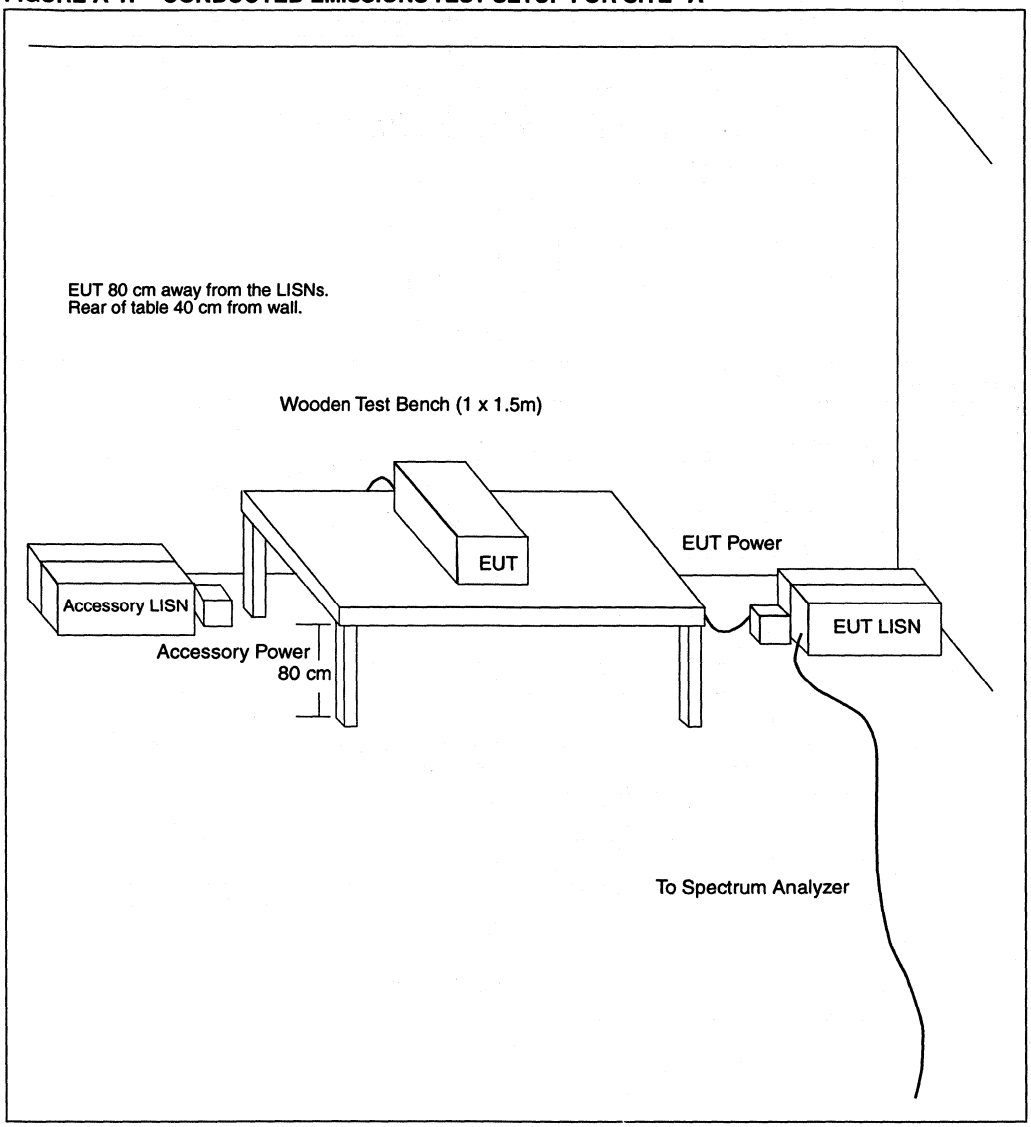
© 1996 Microchip Technology Inc.

CONCLUSION

The PIC16C54A device can be implemented into system designs that are required to be certified to the FCC Class B specification limits as defined by the FCC Title 47, Part 15 Subpart B and IEC 801-3 susceptibility.

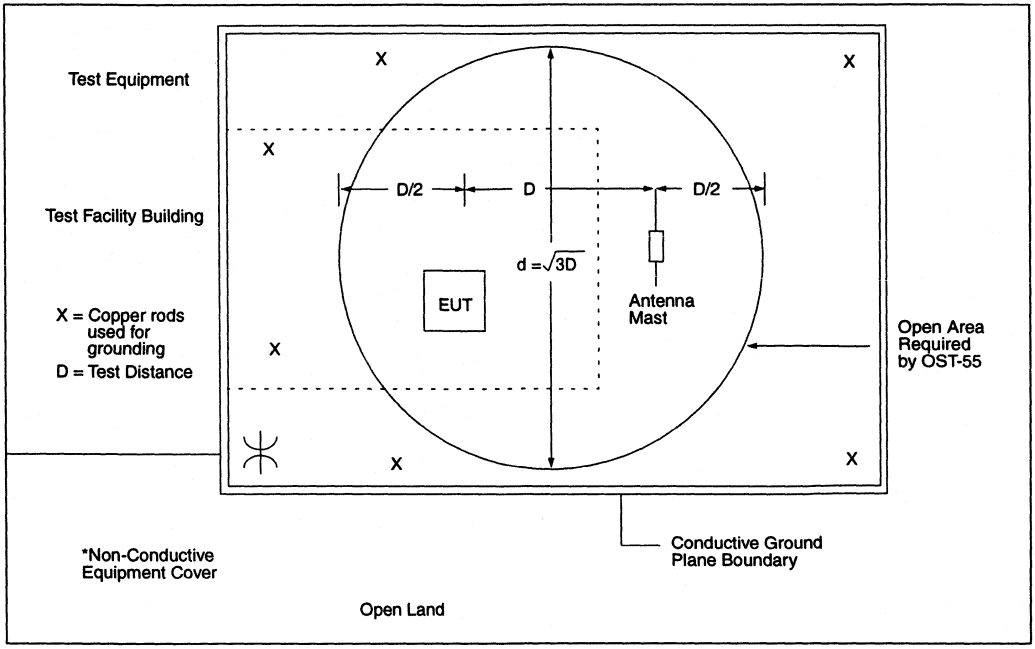
APPENDIX A: APPENDIX A: TEST SETUPS

FIGURE A-1: CONDUCTED EMISSIONS TEST SETUP FOR SITE "A"



© 1996 Microchip Technology Inc.

FIGURE A-2: RADIATED EMISSIONS TEST SETUP FOR SITE "B"



© 1996 Microchip Technology Inc.

NOTES:

© 1996 Microchip Technology Inc.

Implementing Long Calls

Author: *Mark Palmer*
Microchip Technology Inc.

INTRODUCTION

This application note discusses how to implement "long calls" in the PIC16C5X architecture. The use of long call can simplify the partitioning of the application program with minimal software overhead.

In the PIC16C5X architecture, the program memory page size is 512 words. Depending on the device, the program memory may be as large as 2K words (as in the PIC16C57 or PIC16C58 devices). The program counter (and stack) width range from 9- to 11-bits, depending on the amount of program memory. Table 1 shows the width of the Program Counter (PC) and Stack for the various devices.

The low order 8-bits of the program counter are accessible by the user program. These bits are contained in the PC register.

TABLE 1: PC AND STACK WIDTH

DEVICE	Width (bits)		Program Memory (Words)
	Program Counter	Stack	
PIC16C54 / PIC16C55	9	9	512
PIC16C56	10	10	1K
PIC16C57 / PIC16C58	11	11	2K

Figure 1 shows the structure of the PC. Since A8 is forced to 0 by CALL instructions, the start address of subroutines must be in the first 256 words of each program memory page. Depending on the size and number of called subroutines, this limitation may become a burden to the software developer. The implementation of a "long call" eases this, by allowing the subroutine to be anywhere in the program memory page. The three important concepts for understanding the implementation of the long call are:

1. A CALL instruction loads the entire PC onto the Stack.
2. A GOTO instruction does not affect the Stack.
3. A GOTO instruction can branch to any location in a program memory page.

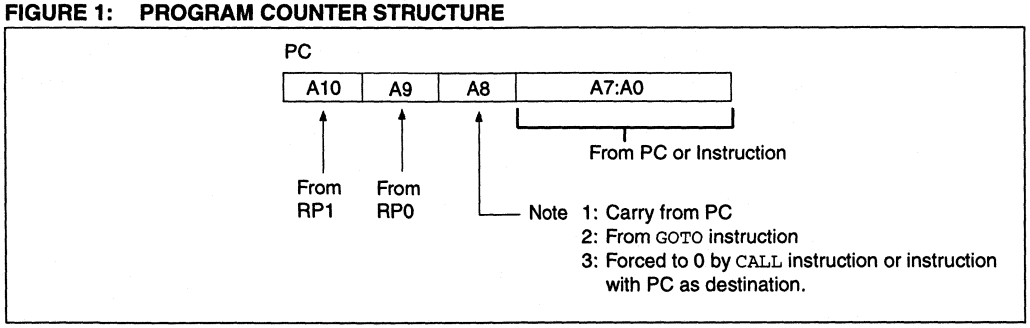
Also to select the desired page, the PA1 and PA0 bits (STATUS<6:5>) must be programmed accordingly. These bits do not get loaded into A10:A9, of the PC, until one of the following occurs:

- A CALL instruction
- A GOTO instruction
- An instruction that modifies the PC register (PC<:7:0>), such as `ADDWF PC,F`.

So a CALL instruction followed by a GOTO instruction will always remain in the same page as the intended call. This allows the developer to place "call vectors" at the first 256 words of each page. The instruction at the "call vector" then executes a GOTO instruction to the subroutine anywhere in that page. The RETLW instruction, of the subroutine, will then POP the stack. The Stack contains the PUSHed PC value from the CALL instruction.

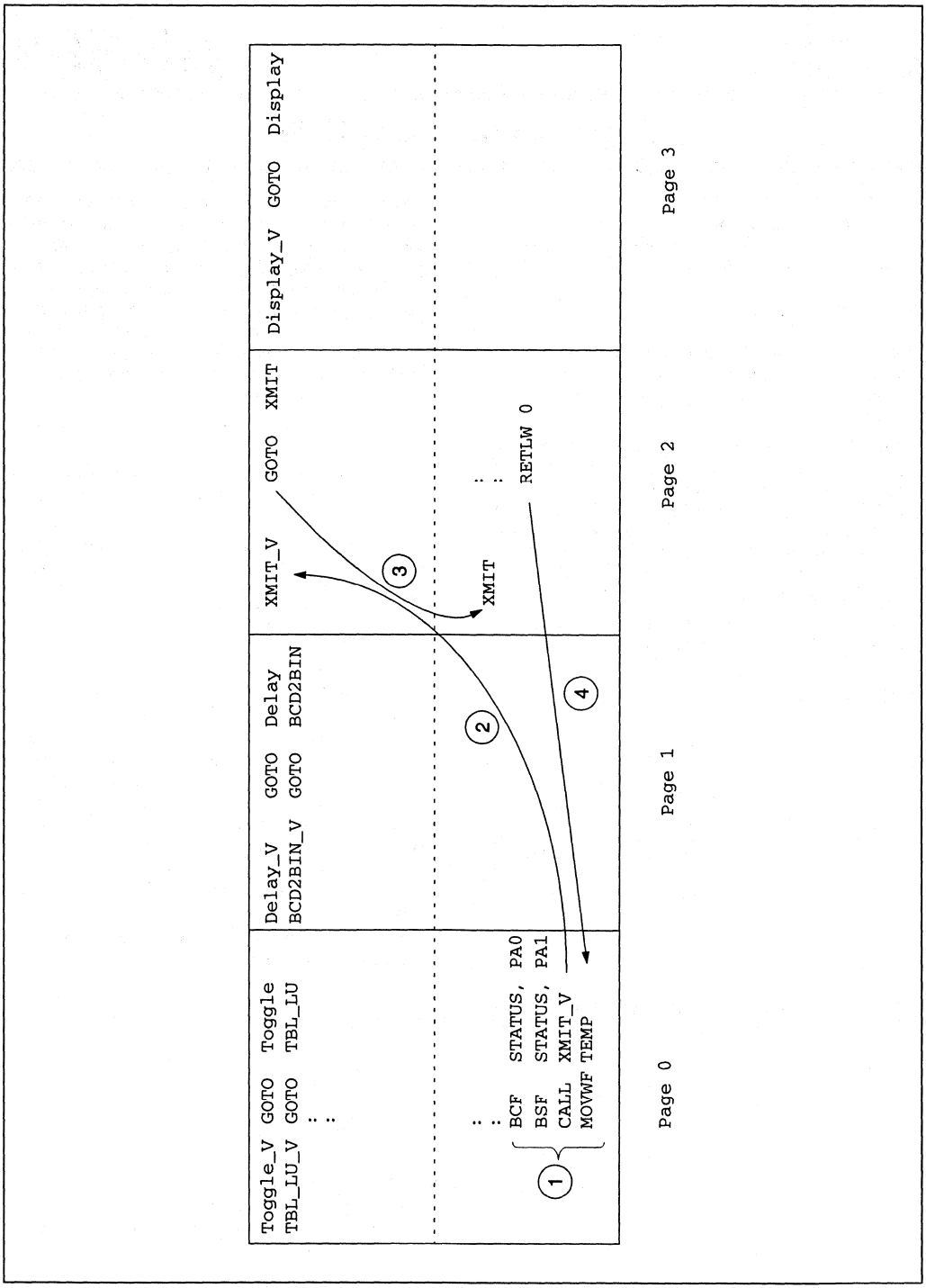
Figure 2 shows an example of a "long call" sequence in a device with 2K-words.

FIGURE 1: PROGRAM COUNTER STRUCTURE



© 1997 Microchip Technology Inc.

FIGURE 2: EXAMPLE OF A "LONG CALL" SEQUENCE



© 1997 Microchip Technology Inc.

The flow that occurs in Figure 2 is as follows:

1. Select the program memory page of the desired subroutine and execute the call to that subroutine.
2. The program loads the Stack with the PC + 1 address, branches to the selected page and specified address of the "call vector" (must be in the first 256 locations of the page).
3. Executes a GOTO instruction, to have access to the entire program memory page. Then executes the subroutine.
4. Executes the RETLW instruction, which POPs the new PC value from the Stack. This causes program execution to continue at the instruction after the CALL instruction.

The use of "long calls" could be used to place all the subroutines in selected page(s), since the entire page can contain the subroutines (not restricted to the top half of the page). The placing of all subroutines in fewer program memory pages can reduce the overhead of specifying the required pages, since they are changed less frequently.

Use of the MPASM assembler can ease in verifying that CALL vectors and the CALL routine are in the same program memory page. Example 1 shows the use of assembler directives to print user defined warning or error messages in the listing file. These are shown as the shaded conditional statements. These messages are only printed in the listing file, and no indication of these messages is shown at the completion of assembly.

© 1997 Microchip Technology Inc.

EXAMPLE 1: USE OF ASSEMBLER DIRECTIVES

```
;
P1_TOP                  EQU         0x0000          ; First address in page 0
P2_TOP                  EQU         0x0200          ; First address in page 1
P3_TOP                  EQU         0x0400          ; First address in page 2
P4_TOP                  EQU         0x0600          ; Reset vector address in page 1
RESET_V                 EQU         0x07FF
;
     org    P1_TOP
;
     :
     :
     :
;
     org    P3_TOP
;
My_Subroutine_V     GOTO    My_Subroutine     ; Vector for My_Subroutine
     :
     :
     :
;
My_Subroutine                                  ; My_Subroutine routine
;
    if ( ( My_Subroutine_V & 0x0600 ) != ( My_Subroutine & 0x0600 ) )
      MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
      endif
;
     :
     :
     :
;
My_Subroutine_END    RETURN
;
    if ( ( My_Subroutine_V & 0x0600) != (My_Subroutine_END & 0x0600) )
      MESSG   "Warning - User Defined: Call routine crosses page boundry"
    endif
;
     :
     :
     :
;
       org   RESET_V                           ; Program memory address for the reset vector
;
                GOTO          START            ; Goto the beginning of the program
```

CONCLUSION

The use of "long calls" may ease the development of application programs. For minimal overhead, the application program can execute a subroutine from anywhere in the program memory, and return to the desired location. This eases the development of the application program by reducing the mapping of subroutine in the first 256 words of each program memory page. The use of "long calls" is possible in any of the PIC16C5X devices, but is most useful in devices with more than one program memory page. For devices with more than one page of program memory, the assembler directives can be used to verify that the subroutines are in the program memory page.

© 1997 Microchip Technology Inc.

Macros for Page and Bank Switching

Author: *Mark Palmer*
Microchip Technology Inc.
Contributions: *Mike Morse*
Sr. Field Applications Engineer
(Dallas)

INTRODUCTION

This application note discusses the use of the MPASM assembler's conditional assembly to automatically switch between program memory pages or to set the data memory banks. These macros, along with the long call technique (Application Note AN581), ease the development of software. Though the use of these macros can simplify program memory paging and data memory banking with minimal software overhead, the use of these macros without thought can cause unnecessary (duplicate) instructions to be used, by setting page or bank bits unnecessarily.

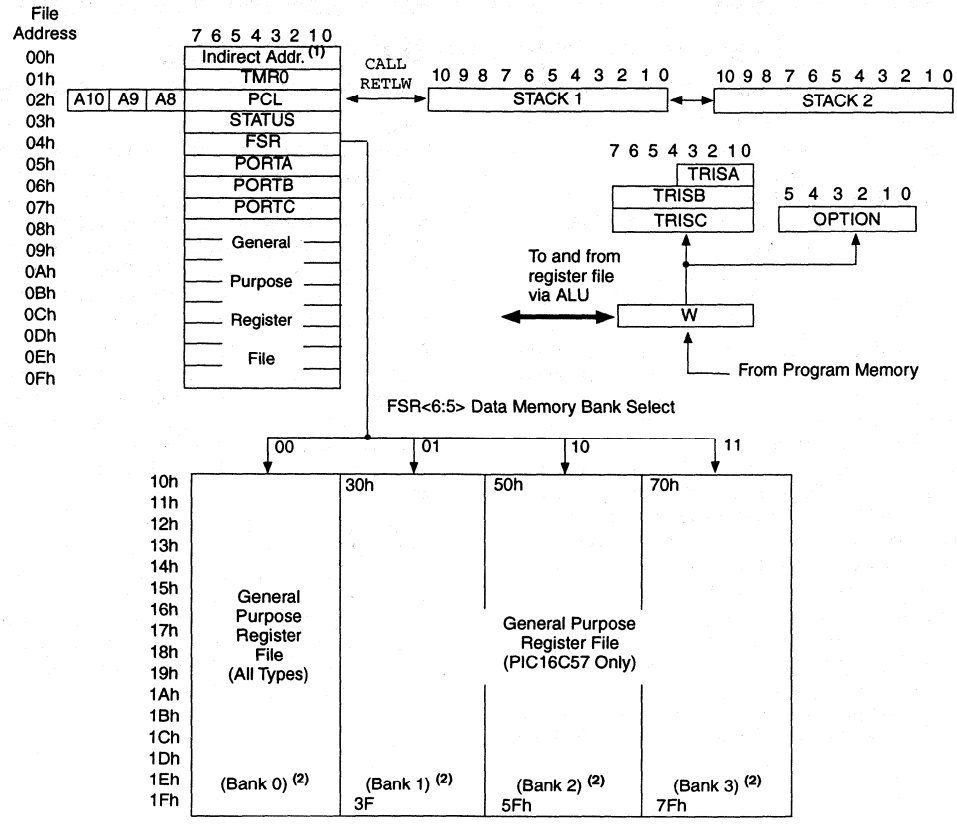
The PIC16C5X family of devices has an architecture in which program memory has up to four pages of program memory (512 words / page) and four banks of data memory (16 bytes / bank). Two bits in the STATUS register, PA1:PA0, are used to manage the program memory pages. Two bits of the FSR register, bits 6 and 5, manage the data memory banks. We will call the FSR<5> bit RP0 and the FSR<6> bit RP1 (for Register Page 0 and 1). The naming of these bits RP1 and RP0 should not be confused with the similarly named bits in the PIC16CXX family (PIC16C64, PIC16C71, etc.). The RP bits for the PIC16CXX family are found in the STATUS register, as opposed to the FSR register for the PIC16C5X family. The use of these macros can be modified to support the PIC16CXX family.

The program memory organization is shown in Figure 1 and the data memory organization is shown in Figure 2. To use the macros for the data memory, the data memory locations must be EQUated for the absolute address, and not the relative address in the bank. The relative address is the lower 5-bits of the data memory address.

When the address of the data memory has the MSb (bit4) of the direct address cleared, or FSR<4> cleared (for indirect addressing), the address 0h through 0Fh is accessed. That is, when accessing addresses 0h through 0Fh the bank selection (FSR<6:5>) bits are ignored. This means that data memory addresses 'xxx0 xxxx'b access the data memory address 0xh (x is anywhere from 0 - F).

© 1997 Microchip Technology Inc.

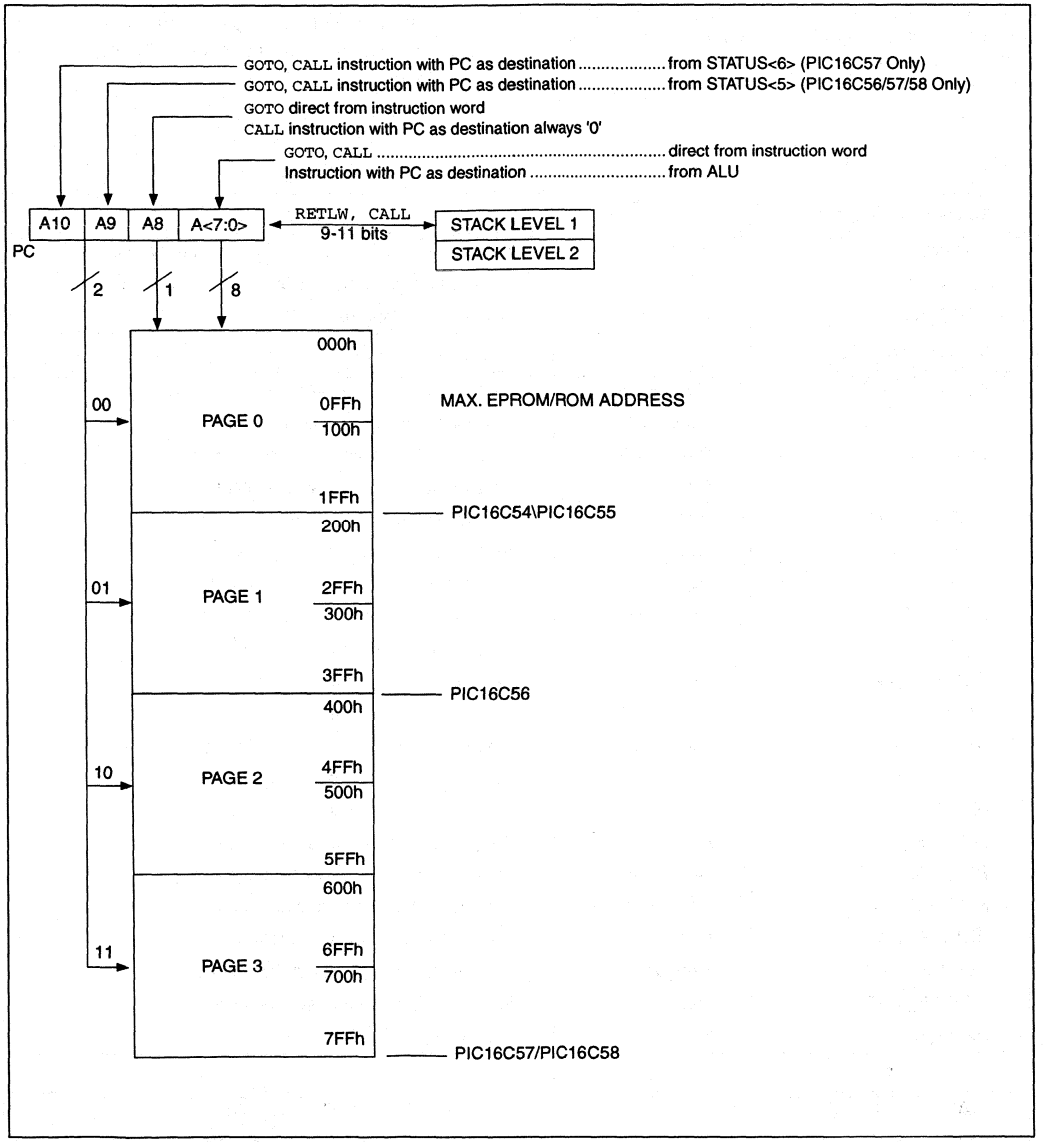
FIGURE 1: PROGRAM MEMORY ORGANIZATION



Note 1: Not a physically implemented register.
2: Bank 0 is available on all microcontrollers while Bank 1, Bank 2, and Bank 3 are only available on the PIC16C57 and PIC16C58.

© 1997 Microchip Technology Inc.

FIGURE 2: DATA MEMORY MAP



© 1997 Microchip Technology Inc.

The use of MPASM's conditional assembly, allows the selection of source code to be assembled based on the address of the symbol / label. The macros supplied are show in Table 1.

They can be grouped into three categories:

1. Configuring of the program memory pages.
2. Configuring of the data memory banks.
3. Other.

TABLE 1: MACROS

Program Calling Paging	Operands	Operation
CALLM	address	Sets page bits, then CALLs the specified routine
GOTOM	address	Sets page bits, then GOTOs the specified address
PAGE_MAC	address	Sets the specified page bits
Data Memory Banking		
ADDWF_MAC	Reg, dest	Sets Bank bits, then executes the ADDWF
ANDWF_MAC	Reg, dest	Sets Bank bits, then executes the ANDWF
BCF_MAC	Reg, bit	Sets Bank bits, then executes the BCF
BSF_MAC	Reg, bit	Sets Bank bits, then executes the BSF
BTFSC_MAC	Reg, bit	Sets Bank bits, then executes the BTFSC
BTFSS_MAC	Reg, bit	Sets Bank bits, then executes the BTFSS
CLRF_MAC	Reg	Sets Bank bits, then executes the CLRF
COMF_MAC	Reg, dest	Sets Bank bits, then executes the COMF
DECF_MAC	Reg, dest	Sets Bank bits, then executes the DECF
DECFSZ_MAC	Reg, dest	Sets Bank bits, then executes the DECFSZ
INCF_MAC	Reg, dest	Sets Bank bits, then executes the INCF
INCFSZ_MAC	Reg, dest	Sets Bank bits, then executes the INCFSZ
IORWF_MAC	Reg, dest	Sets Bank bits, then executes the IORWF
MOVF_MAC	Reg, dest	Sets Bank bits, then executes the MOVF
MOVWF_MAC	Reg	Sets Bank bits, then executes the MOVWF
RLF_MAC	Reg, dest	Sets Bank bits, then executes the RLF
RRF_MAC	Reg, dest	Sets Bank bits, then executes the RRF
SUBWF_MAC	Reg, dest	Sets Bank bits, then executes the SUBWF
SWAPF_MAC	Reg, dest	Sets Bank bits, then executes the SWAPF
XORWF_MAC	Reg, dest	Sets Bank bits, then executes the XORWF
BANK_MAC	Reg	Sets the specified Bank bits
Other		
SAVE_W_STATUS	-	Saves the W and STATUS registers
RESTORE_W_STATUS	-	Restores the W and STATUS registers

© 1997 Microchip Technology Inc.

These macros (Appendix A) ease the development of programs, but care should be taken in their use so that redundant instructions are not caused. An example of this (Example 1) is if you wanted to do the operations, INCF and BTFSS, on data memory location CNTR (in Bank 3) and the FSR was pointing to some other bank. The use of the macros for both operations would cause six program memory locations to be assembled, while with some thought only four words are needed (Example 2).

CONCLUSION

The use of these macros simplify program development by managing the memory resources of the PIC16C5X device. If the application program becomes too large for the device's program memory, it is recommended to study the listing file for any unnecessary code due to non-optimum usage of these macros. The MAC_TST.ASM file, is supplied to show how these macros work in a program.

EXAMPLE 1: GENERATION OF UNNECESSARY CODE

```
INCF_MAC    CNTR, F    ->    BSF     FSR, 5
                             BSF     FSR, 6
                             INCF    CNTR, F
BTFSS_MAC   CNTR, 5    ->    BSF     FSR, 5     ; Unnecessary, already in bank
                             BSF     FSR, 6     ; Unnecessary, already in bank
                             BTFSS   CNTR, 5
```

EXAMPLE 2: GENERATION OF OPTIMUM CODE

```
INCF_MAC    CNTR, F    ->    BSF     FSR, 5
                             BSF     FSR, 6
                             INCF    CNTR, F
BTFSS       CNTR, 5    ->    BTFSS   CNTR, 5
```

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: MACRO FILE

```
MPASM 01.40 Released        LCALL_5X.ASM   1-16-1997  17:13:33         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C57, n = 66
                     00002         ERRORLEVEL  -302, -306
                     00003 ;
                     00004 ;               Program = LCALL_5x.asm
                     00005 ;               Revision Date:  5-07-94
                     00006 ;                               1-15-97    Compatibility with MPASMWIN 1.40
                     00007 ;
                     00008 ;
  00000000           00009 P1_TOP          EQU     0x0000
  000001FF           00010 P1_BOTTOM       EQU     0x01FF
  00000200           00011 P2_TOP          EQU     0x0200
  000003FF           00012 P2_BOTTOM       EQU     0x03FF
  00000400           00013 P3_TOP          EQU     0x0400
  000005FF           00014 P3_BOTTOM       EQU     0x05FF
  00000600           00015 P4_TOP          EQU     0x0600
  000007FF           00016 P4_BOTTOM       EQU     0x07FF
  000007FF           00017 RESET_V         EQU     0x07FF
                     00018 ;
  00000003           00019 STATUS          EQU     0x03        ; Status Register
  00000005           00020 PA0             EQU     0x05        ; Program Memory Page Address bit 0
  00000006           00021 PA1             EQU     0x06        ; Program Memory Page Address bit 1
                     00022 ;
  00000004           00023 FSR             EQU     0x04        ; FSR Register
  00000005           00024 RP0             EQU     0x05        ; Direct Addressing Register Bank bit 0
  00000006           00025 RP1             EQU     0x06        ; Direct Addressing Register Bank bit 1
                     00026 ;
  00000010           00027 BANK0_T         EQU     0x10
  00000030           00028 BANK1_T         EQU     0x30
  00000050           00029 BANK2_T         EQU     0x50
  00000070           00030 BANK3_T         EQU     0x70
                     00031 ;
0000                 00032                 org     P1_TOP
                     00033 ;
0000 0A07            00034 P1_CALL_1_V     GOTO    P1_CALL_1
0001 0A0A            00035 P1_CALL_2_V     GOTO    P1_CALL_2
```


© 1997 Microchip Technology Inc.

```
0002 0A0D          00036 P1_CALL_3_V     GOTO    P1_CALL_3
0003 0A1F          00037 P1_CALL_4_V     GOTO    P1_CALL_4
0004 0A64          00038 P1_CALL_5_V     GOTO    P1_CALL_5
                   00039 ;
                   00040 ;
  00000010         00041 D_address       EQU     0x10
  00000001         00042 F               EQU     1
  00000000         00043 W               EQU     0
                   00044 ;
                   00045      include    <AUTO_PG.MAC>
                   00408      list
                   00409
                   00046 ;
0005 090D          00047 START           CALL    P1_CALL_3
                   00048 ;
0006 0A06          00049 LOOP            GOTO    LCOP
                   00050 ;
                   00051 ;
0007               00052 P1_CALL_1
                   00053 ;
                   00054     if ( (P1_CALL_1_V & 0x0600) != (P1_CALL_1 & 0x0600) )
                   00055        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                   00056     endif
                   00057 ;
0007 0000          00058                 NOP
0008 0000          00059                 NOP
0009 0800          00060 P1_CALL_1_END   RETURN
                   00061 ;
                   00062     if ( (P1_CALL_1 & 0x0600) != (P1_CALL_1_END & 0x0600) )
                   00063        MESSG   "Warning - User Defined: Call routine crosses page boundry"
                   00064     endif
                   00065 ;
                   00066
                   00067
000A               00068 P1_CALL_2
                   00069 ;
                   00070     if ( (P1_CALL_2_V & 0x0600) != (P1_CALL_2 & 0x0600) )
                   00071        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                   00072     endif
                   00073 ;
000A 0000          00074                 NOP
000B 0000          00075                 NOP
000C 0800          00076 P1_CALL_2_END   RETURN
                   00077 ;
                   00078     if ( (P1_CALL_2 & 0x0600) != (P1_CALL_2_END & 0x0600) )
                   00079        MESSG   "Warning - User Defined: Call routine crosses page boundry"
                   00080     endif
```

© 1997 Microchip Technology Inc.

```
                    00081 ;
                    00082
                    00083
000D                00084 P1_CALL_3
                    00085 ;
                    00086     if ( (P1_CALL_3_V & 0x0600) != (P1_CALL_3 & 0x0600) )
                    00087        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                    00088     endif
                    00089 ;
000D 0000           00090                 NOP
000E 0000           00091                 NOP
000F 0800           00092 P1_CALL_3_END   RETURN
                    00093 ;
                    00094     if ( (P1_CALL_3 & 0x0600) != (P1_CALL_3_END & 0x0600) )
                    00095        MESSG   "Warning - User Defined: Call routine crosses page boundry"
                    00096     endif
                    00097 ;
                    00098
                    00099     CALLM       P2_CALL_3_V
                        M ;
                        M     if ( ( P2_CALL_3_V & 0x0200 ) == 0x0200 )
0010 05A3               M                 BSF     STATUS, PA0       ; Set PA0 for Program Memory Page
                        M     else
                        M                 BCF     STATUS, PA0       ; Clear PA0 for Program Memory Page
                        M     endif
                        M ;
                        M     if ( ( P2_CALL_3_V & 0x0400 ) == 0x0400 )
                        M                 BSF     STATUS, PA1       ; Set PA1 for Program Memory Page
                        M     else
0011 04C3               M                 BCF     STATUS, PA1       ; Clear PA1 for Program Memory Page
                        M     endif
                        M ;
0012 0905               M                 CALL    P2_CALL_3_V
0013 0000           00100                 nop
0014 0000           00101                 nop
                    00102 ;
                    00103           CALLM       P4_CALL_2_V
                        M ;
                        M      if ( ( P4_CALL_2_V & 0x0200 ) == 0x0200 )
0015 05A3               M                 BSF     STATUS, PA0       ; Set PA0 for Program Memory Page
                        M      else
                        M                 BCF     STATUS, PA0       ; Clear PA0 for Program Memory Page
                        M      endif
                        M ;
                        M      if ( ( P4_CALL_2_V & 0x0400 ) == 0x0400 )
0016 05C3               M                 BSF     STATUS, PA1       ; Set PA1 for Program Memory Page
                        M      else
```


© 1997 Microchip Technology Inc.

```
                        M                BCF     STATUS, PA1        ; Clear PA1 for Program Memory Page
                        M       endif
                        M ;
0017 0901               M                CALL    P4_CALL_2_V
0018 0000          00104                 nop
0019 0000          00105                 nop
                   00106 ;
                   00107             CALLM       P3_CALL_1_V
                        M ;
                        M       if ( ( P3_CALL_1_V & 0x0200 ) == 0x0200 )
                        M                BSF     STATUS, PA0        ; Set PA0 for Program Memory Page
                        M       else
001A 04A3               M                BCF     STATUS, PA0        ; Clear PA0 for Program Memory Page
                        M       endif
                        M ;
                        M       if ( ( P3_CALL_1_V & 0x0400 ) == 0x0400 )
001B 05C3               M                BSF     STATUS, PA1        ; Set PA1 for Program Memory Page
                        M       else
                        M                BCF     STATUS, PA1        ; Clear PA1 for Program Memory Page
                        M       endif
                        M ;
001C 0900               M                CALL    P3_CALL_1_V
001D 0000          00108                 nop
001E 0000          00109                 nop
                   00110
                   00111 ;
001F               00112 P1_CALL_4
                   00113 ;
                   00114     if ( (P1_CALL_4_V & 0x0600) != (P1_CALL_4 & 0x0600) )
                   00115        MESSG    "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                   00116     endif
                   00117 ;
001F 0000          00118                  NOP
0020 0000          00119                  NOP
0021 0800          00120 P1_CALL_4_END    RETURN
                   00121 ;
                   00122     if ( (P1_CALL_4 & 0x0600) != (P1_CALL_4_END & 0x0600) )
                   00123        MESSG    "Warning - User Defined: Call routine crosses page boundry"
                   00124     endif
                   00125 ;
0022 0000          00126                 nop
0023 0000          00127                 nop
                   00128
                   00129                 ADDWF_MAC      D_address, F
                        M ;
                        M       if ( ( D_address & 0x020 ) == 0x020 )
                        M                BSF     FSR, RP0           ; Set RP0 for Data Memory Page
```

© 1997 Microchip Technology Inc.

```
                    M       else
0024 04A4           M               BCF     FSR, RP0        ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( D_address & 0x040 ) == 0x040 )
                    M               BSF     FSR, RP1        ; Set RP1 for Data Memory Bank
                    M       else
0025 04C4           M               BCF     FSR, RP1        ; Clear RP1 for Data Memory Bank
                    M       endif
0026 01F0           M               ADDWF   D_address, F
                  00130             ANDWF_MAC       D_address, F
                    M ;
                    M       if ( ( D_address & 0x020 ) == 0x020 )
                    M               BSF     FSR, RP0        ; Set RP0 for Data Memory Bank
                    M       else
0027 04A4           M               BCF     FSR, RP0        ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( D_address & 0x040 ) == 0x040 )
                    M               BSF     FSR, RP1        ; Set RP1 for Data Memory Bank
                    M       else
0028 04C4           M               BCF     FSR, RP1        ; Clear RP1 for Data Memory Bank
                    M       endif
0029 0170           M               ANDWF   D_address, F
                  00131             BCF_MAC         D_address, 7
                    M ;
                    M       if ( ( D_address & 0x020 ) == 0x020 )
                    M               BSF     FSR, RP0        ; Set RP0 for Data Memory Bank
                    M       else
002A 04A4           M               BCF     FSR, RP0        ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( D_address & 0x040 ) == 0x040 )
                    M               BSF     FSR, RP1        ; Set RP1 for Data Memory Bank
                    M       else
002B 04C4           M               BCF     FSR, RP1        ; Clear RP1 for Data Memory Bank
                    M       endif
002C 04F0           M               BCF     D_address, 7
                  00132             BSF_MAC         D_address, 7
                    M ;
                    M       if ( ( D_address & 0x020 ) == 0x020 )
                    M               BSF     FSR, RP0        ; Set RP0 for Data Memory Bank
                    M       else
002D 04A4           M               BCF     FSR, RP0        ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( D_address & 0x040 ) == 0x040 )
```


© 1997 Microchip Technology Inc.

```
                    M                   BSF     FSR, RP1            ; Set RP1 for Data Memory Bank
                    M       else
002E 04C4           M                   BCF     FSR, RP1            ; Clear RP1 for Data Memory Bank
                    M       endif
002F 05F0           M                   BSF     D_address, 7
                  00133                 BTFSC_MAC       D_address, 7
                    M ;
                    M       if ( ( D_address & 0x020 ) == 0x020 )
                    M                   BSF     FSR, RP0            ; Set RP0 for Data Memory Bank
                    M       else
0030 04A4           M                   BCF     FSR, RP0            ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( D_address & 0x040 ) == 0x040 )
                    M                   BSF     FSR, RP1            ; Set RP1 for Data Memory Bank
                    M       else
0031 04C4           M                   BCF     FSR, RP1            ; Clear RP1 for Data Memory Bank
                    M       endif
0032 06F0           M                   BTFSC   D_address, 7
                  00134                 BTFSS_MAC       D_address, 7
                    M ;
                    M       if ( ( D_address & 0x020 ) == 0x020 )
                    M                   BSF     FSR, RP0            ; Set RP0 for Data Memory Bank
                    M       else
0033 04A4           M                   BCF     FSR, RP0            ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( D_address & 0x040 ) == 0x040 )
                    M                   BSF     FSR, RP1            ; Set RP1 for Data Memory Bank
                    M       else
0034 04C4           M                   BCF     FSR, RP1            ; Clear RP1 for Data Memory Bank
                    M       endif
0035 07F0           M                   BTFSS   D_address, 7
                  00135                 CLRF_MAC        D_address
                    M ;
                    M       if ( ( D_address & 0x020 ) == 0x020 )
                    M                   BSF     FSR, RP0            ; Set RP0 for Data Memory Bank
                    M       else
0036 04A4           M                   BCF     FSR, RP0            ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( D_address & 0x040 ) == 0x040 )
                    M                   BSF     FSR, RP1            ; Set RP1 for Data Memory Bank
                    M       else
0037 04C4           M                   BCF     FSR, RP1            ; Clear RP1 for Data Memory Bank
                    M       endif
0038 0070           M                   CLRF    D_address
```

© 1997 Microchip Technology Inc.

```
                    00136                COMF_MAC       D_address, F
                        M ;
                        M       if ( ( D_address & 0x020 ) == 0x020 )
                        M               BSF     FSR, RP0          ; Set RP0 for Data Memory Bank
                        M       else
0039 04A4               M               BCF     FSR, RP0          ; Clear RP0 for Data Memory Bank
                        M       endif
                        M ;
                        M       if ( ( D_address & 0x040 ) == 0x040 )
                        M               BSF     FSR, RP1          ; Set RP1 for Data Memory Bank
                        M       else
003A 04C4               M               BCF     FSR, RP1          ; Clear RP1 for Data Memory Bank
                        M       endif
003B 0270               M               COMF    D_address, F
                    00137               DECF_MAC       D_address, F
                        M ;
                        M       if ( ( D_address & 0x020 ) == 0x020 )
                        M               BSF     FSR, RP0          ; Set RP0 for Data Memory Bank
                        M       else
003C 04A4               M               BCF     FSR, RP0          ; Clear RP0 for Data Memory Bank
                        M       endif
                        M ;
                        M       if ( ( D_address & 0x040 ) == 0x040 )
                        M               BSF     FSR, RP1          ; Set RP1 for Data Memory Bank
                        M       else
003D 04C4               M               BCF     FSR, RP1          ; Clear RP1 for Data Memory Bank
                        M       endif
003E 00F0               M               DECF    D_address, F
                    00138               DECFSZ_MAC     D_address, F
                        M ;
                        M       if ( ( D_address & 0x020 ) == 0x020 )
                        M               BSF     FSR, RP0          ; Set RP0 for Data Memory Bank
                        M       else
003F 04A4               M               BCF     FSR, RP0          ; Clear RP0 for Data Memory Bank
                        M       endif
                        M ;
                        M       if ( ( D_address & 0x040 ) == 0x040 )
                        M               BSF     FSR, RP1          ; Set RP1 for Data Memory Bank
                        M       else
0040 04C4               M               BCF     FSR, RP1          ; Clear RP1 for Data Memory Bank
                        M       endif
0041 02F0               M               DECFSZ  D_address, F
                    00139               INCF_MAC       D_address, F
                        M ;
                        M       if ( ( D_address & 0x020 ) == 0x020 )
                        M               BSF     FSR, RP0          ; Set RP0 for Data Memory Bank
                        M       else
```


© 1997 Microchip Technology Inc.

```
0042 04A4                M                BCF     FSR, RP0          ; Clear RP0 for Data Memory Bank
                         M        endif
                         M ;
                         M        if ( ( D_address & 0x040 ) == 0x040 )
                         M                BSF     FSR, RP1          ; Set RP1 for Data Memory Bank
                         M        else
0043 04C4                M                BCF     FSR, RP1          ; Clear RP1 for Data Memory Bank
                         M        endif
0044 02B0                M                INCF    D_address, F
                       00140              INCFSZ_MAC      D_address, F
                         M ;
                         M        if ( ( D_address & 0x020 ) == 0x020 )
                         M                BSF     FSR, RP0          ; Set RP0 for Data Memory Bank
                         M        else
0045 04A4                M                BCF     FSR, RP0          ; Clear RP0 for Data Memory Bank
                         M        endif
                         M ;
                         M        if ( ( D_address & 0x040 ) == 0x040 )
                         M                BSF     FSR, RP1          ; Set RP1 for Data Memory Bank
                         M        else
0046 04C4                M                BCF     FSR, RP1          ; Clear RP1 for Data Memory Bank
                         M        endif
0047 03F0                M                INCFSZ  D_address, F
                       00141              IORWF_MAC       D_address, F
                         M ;
                         M        if ( ( D_address & 0x020 ) == 0x020 )
                         M                BSF     FSR, RP0          ; Set RP0 for Data Memory Bank
                         M        else
0048 04A4                M                BCF     FSR, RP0          ; Clear RP0 for Data Memory Bank
                         M        endif
                         M ;
                         M        if ( ( D_address & 0x040 ) == 0x040 )
                         M                BSF     FSR, RP1          ; Set RP1 for Data Memory Bank
                         M        else
0049 04C4                M                BCF     FSR, RP1          ; Clear RP1 for Data Memory Bank
                         M        endif
004A 0130                M                IORWF    D_address, F
                       00142              MOVF_MAC        D_address, F
                         M ;
                         M        if ( ( D_address & 0x020 ) == 0x020 )
                         M                BSF     FSR, RP0          ; Set RP0 for Data Memory Bank
                         M        else
004B 04A4                M                BCF     FSR, RP0          ; Clear RP0 for Data Memory Bank
                         M        endif
                         M ;
                         M        if ( ( D_address & 0x040 ) == 0x040 )
                         M                BSF     FSR, RP1          ; Set RP1 for Data Memory Bank
```

© 1997 Microchip Technology Inc.

```
                    M       else
004C 04C4           M               BCF     FSR, RP1        ; Clear RP1 for Data Memory Bank
                    M       endif
004D 0230           M               MOVF    D_address, F
                  00143             MOVWF_MAC       D_address
                    M ;
                    M       if ( ( D_address & 0x020 ) == 0x020 )
                    M               BSF     FSR, RP0        ; Set RP0 for Data Memory Bank
                    M       else
004E 04A4           M               BCF     FSR, RP0        ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( D_address & 0x040 ) == 0x040 )
                    M               BSF     FSR, RP1        ; Set RP1 for Data Memory Bank
                    M       else
004F 04C4           M               BCF     FSR, RP1        ; Clear RP1 for Data Memory Bank
                    M       endif
0050 0030           M               MOVWF    D_address
                  00144             RLF_MAC         D_address, F
                    M ;
                    M       if ( ( D_address & 0x020 ) == 0x020 )
                    M               BSF     FSR, RP0        ; Set RP0 for Data Memory Bank
                    M       else
0051 04A4           M               BCF     FSR, RP0        ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( D_address & 0x040 ) == 0x040 )
                    M               BSF     FSR, RP1        ; Set RP1 for Data Memory Bank
                    M       else
0052 04C4           M               BCF     FSR, RP1        ; Clear RP1 for Data Memory Bank
                    M       endif
0053 0370           M               RLF     D_address, F
                  00145             RRF_MAC         D_address, F
                    M ;
                    M       if ( ( D_address & 0x020 ) == 0x020 )
                    M               BSF     FSR, RP0        ; Set RP0 for Data Memory Bank
                    M       else
0054 04A4           M               BCF     FSR, RP0        ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( D_address & 0x040 ) == 0x040 )
                    M               BSF     FSR, RP1        ; Set RP1 for Data Memory Bank
                    M       else
0055 04C4           M               BCF     FSR, RP1        ; Clear RP1 for Data Memory Bank
                    M       endif
0056 0330           M               RRF     D_address, F
                  00146             SUBWF_MAC       D_address, F
```


© 1997 Microchip Technology Inc.

```
                    M ;
                    M       if ( ( D_address & 0x020 ) == 0x020 )
                    M               BSF     FSR, RP0          ; Set RP0 for Data Memory Bank
                    M       else
0057 04A4           M               BCF     FSR, RP0          ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( D_address & 0x040 ) == 0x040 )
                    M               BSF     FSR, RP1          ; Set RP1 for Data Memory Bank
                    M       else
0058 04C4           M               BCF     FSR, RP1          ; Clear RP1 for Data Memory Bank
                    M       endif
0059 00B0           M               SUBWF   D_address, F
                  00147             SWAPF_MAC       D_address, F
                    M ;
                    M       if ( ( D_address & 0x020 ) == 0x020 )
                    M               BSF     FSR, RP0          ; Set RP0 for Data Memory Bank
                    M       else
005A 04A4           M               BCF     FSR, RP0          ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( D_address & 0x040 ) == 0x040 )
                    M               BSF     FSR, RP1          ; Set RP1 for Data Memory Bank
                    M       else
005B 04C4           M               BCF     FSR, RP1          ; Clear RP1 for Data Memory Bank
                    M       endif
005C 03B0           M               SWAPF   D_address, F
                  00148             XORWF_MAC       D_address, F
                    M ;
                    M       if ( ( D_address & 0x020 ) == 0x020 )
                    M               BSF     FSR, RP0          ; Set RP0 for Data Memory Bank
                    M       else
005D 04A4           M               BCF     FSR, RP0          ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( D_address & 0x040 ) == 0x040 )
                    M               BSF     FSR, RP1          ; Set RP1 for Data Memory Bank
                    M       else
005E 04C4           M               BCF     FSR, RP1          ; Clear RP1 for Data Memory Bank
                    M       endif
005F 01B0           M               XORWF   D_address, F
                  00149             PAGE_MAC        P1_CALL_4
                    M ;
                    M       if ( ( P1_CALL_4 & 0x0200 ) == 0x0200 )
                    M               BSF     STATUS, PA0       ; Set PA0 for Program Memory Page
                    M       else
0060 04A3           M               BCF     STATUS, PA0       ; Clear PA0 for Program Memory Page
```

© 1997 Microchip Technology Inc.

```
                          M      endif
                          M ;
                          M      if ( ( P1_CALL_4 & 0x0400 ) == 0x0400 )
                          M              BSF     STATUS, PA1       ; Set PA1 for Program Memory Page
                          M      else
0061 04C3                 M              BCF     STATUS, PA1       ; Clear PA1 for Program Memory Page
                          M      endif
                          M ;
                        00150            BANK_MAC        D_address
                          M ;
                          M      if ( ( D_address & 0x020 ) == 0x020 )
                          M              BSF     FSR, RP0          ; Set RP0 for Data Memory Bank
                          M      else
0062 04A4                 M              BCF     FSR, RP0          ; Clear RP0 for Data Memory Bank
                          M      endif
                          M ;
                          M      if ( ( D_address & 0x040 ) == 0x040 )
                          M              BSF     FSR, RP1          ; Set RP1 for Data Memory Bank
                          M      else
0063 04C4                 M              BCF     FSR, RP1          ; Clear RP1 for Data Memory Bank
                          M      endif
                        00151
                        00152
                        00153
0064                    00154 P1_CALL_5
                        00155 ;
                        00156      if ( (P1_CALL_5_V & 0x0600) != (P1_CALL_5 & 0x0600) )
                        00157         MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                        00158      endif
                        00159 ;
0064 0000               00160              NOP
0065 0000               00161              NOP
                        00162 ;
0200                    00163              org     P2_TOP       ; This is to force an intentional User Defined Message
                        00164 ;
0200 0000               00165              NOP
0201 0000               00166              NOP
0202 0800               00167 P1_CALL_5_END  RETURN
                        00168 ;
                        00169      if ( (P1_CALL_5 & 0x0600) != (P1_CALL_5_END & 0x0600) )
Message[301]: MESSAGE: (Warning - User Defined: Call routine crosses page boundry)
                        00170         MESSG   "Warning - User Defined: Call routine crosses page boundry"
                        00171      endif
                        00172 ;
                        00173
0203 0A08               00174 P2_CALL_1_V    GOTO    P2_CALL_1
0204 0A0B               00175 P2_CALL_2_V    GOTO    P2_CALL_2
```


© 1997 Microchip Technology Inc.

```
0205 0A13          00176 P2_CALL_3_V    GOTO    P2_CALL_3
0206 0A19          00177 P2_CALL_4_V    GOTO    P2_CALL_4
0207 0A1C          00178 P2_CALL_5_V    GOTO    P2_CALL_5
                   00179
                   00180
0208 0000          00181 P2_CALL_1      NOP
0209 0000          00182                NOP
020A 0800          00183                RETURN
                   00184 ;
                   00185     if ( (P2_CALL_1_V & 0x0600) != (P2_CALL_1 & 0x0600) )
                   00186        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                   00187     endif
                   00188
                   00189 ;
020B 0000          00190 P2_CALL_2      NOP
020C 0000          00191                NOP
020D 0800          00192                RETURN
                   00193 ;
                   00194     if ( (P2_CALL_2_V & 0x0600) != (P2_CALL_2 & 0x0600) )
                   00195        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                   00196     endif
                   00197
                   00198 ;
                   00199                CALLM   P1_CALL_3_V
                       M ;
                       M      if ( ( P1_CALL_3_V & 0x0200 ) == 0x0200 )
                       M                BSF     STATUS, PA0       ; Set PA0 for Program Memory Page
                       M      else
020E 04A3              M                BCF     STATUS, PA0       ; Clear PA0 for Program Memory Page
                       M      endif
                       M ;
                       M      if ( ( P1_CALL_3_V & 0x0400 ) == 0x0400 )
                       M                BSF     STATUS, PA1       ; Set PA1 for Program Memory Page
                       M      else
020F 04C3              M                BCF     STATUS, PA1       ; Clear PA1 for Program Memory Page
                       M      endif
                       M ;
0210 0902              M                CALL    P1_CALL_3_V
0211 0000          00200                nop
0212 0000          00201                nop
                   00202
0213 0000          00203 P2_CALL_3      NOP
0214 0000          00204                NOP
0215 0800          00205                RETURN
                   00206 ;
                   00207     if ( (P2_CALL_3_V & 0x0600) != (P2_CALL_3 & 0x0600) )
                   00208        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
```

© 1997 Microchip Technology Inc.

```
                00209     endif
                00210
                00211 ;
                00212               GOTOM       P1_CALL_2
                    M ;
                    M       if ( ( P1_CALL_2 & 0x0200 ) == 0x0200 )
                    M               BSF     STATUS, PA0         ; Set PA0 for Program Memory Page
                    M       else
0216 04A3           M               BCF     STATUS, PA0         ; Clear PA0 for Program Memory Page
                    M       endif
                    M ;
                    M       if ( ( P1_CALL_2 & 0x0400 ) == 0x0400 )
                    M               BSF     STATUS, PA1         ; Set PA1 for Program Memory Page
                    M       else
0217 04C3           M               BCF     STATUS, PA1         ; Clear PA1 for Program Memory Page
                    M       endif
                    M ;
0218 0A0A           M               GOTO    P1_CALL_2
                00213
                00214 ;
0219 0000       00215 P2_CALL_4     NOP
021A 0000       00216               NOP
021B 0800       00217               RETURN
                00218 ;
                00219     if ( (P2_CALL_4_V & 0x0600) != (P2_CALL_4 & 0x0600) )
                00220        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                00221     endif
                00222
                00223 ;
021C 0000       00224 P2_CALL_5     NOP
021D 0000       00225               NOP
021E 0800       00226               RETURN
                00227 ;
                00228     if ( (P2_CALL_5_V & 0x0600) != (P2_CALL_5 & 0x0600) )
                00229        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                00230     endif
                00231
                00232 ;
                00233         MOVF_MAC    BANK0_T, 0
                    M ;
                    M       if ( ( BANK0_T & 0x020 ) == 0x020 )
                    M               BSF     FSR, RP0            ; Set RP0 for Data Memory Bank
                    M       else
021F 04A4           M               BCF     FSR, RP0            ; Clear RP0 for Data Memory Bank
                    M       endif
                    M ;
                    M       if ( ( BANK0_T & 0x040 ) == 0x040 )
```


© 1997 Microchip Technology Inc.

```
                        M               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
                        M       else
0220 04C4               M               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
                        M       endif
0221 0210               M               MOVF    BANK0_T, 0
                      00234         MOVWF_MAC     BANK1_T
                        M ;
                        M       if ( ( BANK1_T & 0x020 ) == 0x020 )
0222 05A4               M               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
                        M       else
                        M               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
                        M       endif
                        M ;
                        M       if ( ( BANK1_T & 0x040 ) == 0x040 )
                        M               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
                        M       else
0223 04C4               M               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
                        M       endif
0224 0030               M               MOVWF    BANK1_T
                      00235         MOVF_MAC      BANK2_T, 0
                        M ;
                        M       if ( ( BANK2_T & 0x020 ) == 0x020 )
                        M               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
                        M       else
0225 04A4               M               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
                        M       endif
                        M ;
                        M       if ( ( BANK2_T & 0x040 ) == 0x040 )
0226 05C4               M               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
                        M       else
                        M               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
                        M       endif
0227 0210               M               MOVF    BANK2_T, 0
                      00236         MOVWF_MAC     BANK3_T
                        M ;
                        M       if ( ( BANK3_T & 0x020 ) == 0x020 )
0228 05A4               M               BSF     FSR, RP0          ; Set RP0 for Data Memory Page
                        M       else
                        M               BCF     FSR, RP0          ; Clear RP0 for Data Memory Page
                        M       endif
                        M ;
                        M       if ( ( BANK3_T & 0x040 ) == 0x040 )
0229 05C4               M               BSF     FSR, RP1          ; Set RP1 for Data Memory Page
                        M       else
                        M               BCF     FSR, RP1          ; Clear RP1 for Data Memory Page
                        M       endif
022A 0030               M               MOVWF    BANK3_T
```

© 1997 Microchip Technology Inc.

```
                    00237
                    00238 ;
0400                00239               org     P3_TOP
                    00240 ;
0400 0A05           00241 P3_CALL_1_V   GOTO    P3_CALL_1
0401 0A08           00242 P3_CALL_2_V   GOTO    P3_CALL_2
0402 0A0B           00243 P3_CALL_3_V   GOTO    P3_CALL_3
0403 0A0E           00244 P3_CALL_4_V   GOTO    P3_CALL_4
0404 0A05           00245 P3_CALL_5_V   GOTO    P3_CALL_5
                    00246
                    00247
0405 0000           00248 P3_CALL_1     NOP
0406 0000           00249               NOP
0407 0800           00250               RETURN
                    00251 ;
                    00252     if ( (P3_CALL_1_V & 0x0600) != (P3_CALL_1 & 0x0600) )
                    00253        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                    00254     endif
                    00255
                    00256 ;
0408 0000           00257 P3_CALL_2     NOP
0409 0000           00258               NOP
040A 0800           00259               RETURN
                    00260 ;
                    00261     if ( (P3_CALL_2_V & 0x0600) != (P3_CALL_2 & 0x0600) )
                    00262        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                    00263     endif
                    00264
                    00265 ;
040B 0000           00266 P3_CALL_3     NOP
040C 0000           00267               NOP
040D 0800           00268               RETURN
                    00269 ;
                    00270     if ( (P3_CALL_3_V & 0x0600) != (P3_CALL_3 & 0x0600) )
                    00271        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                    00272     endif
                    00273
                    00274 ;
040E 0000           00275 P3_CALL_4     NOP
040F 0000           00276               NOP
0410 0800           00277               RETURN
                    00278 ;
                    00279     if ( (P3_CALL_4_V & 0x0600) != (P3_CALL_4 & 0x0600) )
                    00280        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                    00281     endif
                    00282
                    00283 ;
```


© 1997 Microchip Technology Inc.

```
0600                00284              org     P4_TOP
                    00285 ;
0600 0A08           00286 P4_CALL_1_V  GOTO    P4_CALL_1
0601 0A0B           00287 P4_CALL_2_V  GOTO    P4_CALL_2
0602 0A0E           00288 P4_CALL_3_V  GOTO    P4_CALL_3
0603 0A11           00289 P4_CALL_4_V  GOTO    P4_CALL_4
0604 0A14           00290 P4_CALL_5_V  GOTO    P4_CALL_5
                    00291
                    00292 ;
0605 0000           00293 P3_CALL_5    NOP             ; This is to force an intentional User Defined Message
0606 0000           00294              NOP
0607 0800           00295              RETURN
                    00296 ;
                    00297     if ( (P3_CALL_5_V & 0x0600) != (P3_CALL_5 & 0x0600) )
Message[301]: MESSAGE: (ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page)
                    00298        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                    00299     endif
                    00300
                    00301 ;
0608 0000           00302 P4_CALL_1    NOP
0609 0000           00303              NOP
060A 0800           00304              RETURN
                    00305 ;
                    00306     if ( (P4_CALL_1_V & 0x0600) != (P4_CALL_1 & 0x0600) )
                    00307        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                    00308     endif
                    00309
                    00310 ;
060B 0000           00311 P4_CALL_2    NOP
060C 0000           00312              NOP
060D 0800           00313              RETURN
                    00314 ;
                    00315     if ( (P4_CALL_2_V & 0x0600) != (P4_CALL_2 & 0x0600) )
                    00316        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                    00317     endif
                    00318
                    00319 ;
060E 0000           00320 P4_CALL_3    NOP
060F 0000           00321              NOP
0610 0800           00322              RETURN
                    00323 ;
                    00324     if ( (P4_CALL_3_V & 0x0600) != (P4_CALL_3 & 0x0600) )
                    00325        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                    00326     endif
                    00327
                    00328 ;
0611 0000           00329 P4_CALL_4    NOP
```

© 1997 Microchip Technology Inc.

```
0612 0000           00330              NOP
0613 0800           00331              RETURN
                    00332 ;
                    00333     if ( (P4_CALL_4_V & 0x0600) != (P4_CALL_4 & 0x0600) )
                    00334        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                    00335     endif
                    00336
                    00337 ;
0614 0000           00338 P4_CALL_5    NOP
0615 0000           00339              NOP
0616 0800           00340              RETURN
                    00341 ;
                    00342     if ( (P4_CALL_5_V & 0x0600) != (P4_CALL_5 & 0x0600) )
                    00343        MESSG   "ERROR - User Defined: CALL VECTOR and CALL routine NOT in same page"
                    00344     endif
                    00345 ;
                    00346 ;
07FF                00347              org     RESET_V
                    00348 ;
07FF 0A05           00349              GOTO    START           ; Goto the begining of the program
                    00350
                    00351         end

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXX---------- ----------------
0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXX----- ----------------
0400 : XXXXXXXXXXXXXXXX X--------------- ---------------- ----------------
0600 : XXXXXXXXXXXXXXXX XXXXXXX--------- ---------------- ----------------
07C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   186
Program Memory Words Free:  1862


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     2 reported,    10 suppressed
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN590

A Clock Design Using the PIC16C54 for LED Displays and Switch Inputs

Author: *Dan Matthews*
Microchip Technology Inc.

INTRODUCTION

The purpose of this application note is to design a clock while multiplexing the features as much as possible, allowing the circuit to use the 18-pin PIC16C54. Other devices in the Microchip line expand on this part, making it a good starting point for learning the basics. This design is useful because it utilizes every pin for output and switches some of them to inputs briefly to read the keys. For a more extensive clock design, consult application note AN529.

THE DESIGN

This design is a simple time-of-day clock incorporating four seven-segment LED displays and three input switches. There is also an additional reset switch that would not normally be incorporated into the final design. The schematic is illustrated in Figure 1.

CONNECTIONS

The individual segments of each display are connected together, A-A-A-A, B-B-B-B, etc. The displays are numbered from the right, or least significant digit. The second display from the right is flipped upside down to align its decimal with the third display, creating the center clock colon. Therefore the segments are not tied together evenly straight across on the board, but must compensate for the change in one display's orientation. The common cathode for each display is turned on with transistors connected to the four I/O lines of PORTA. The connections are RA0-CC4/Digit4, RA1-CC3/Digit3, RA2-CC2/Digit2, RA3-CC1/Digit1. A low output turns on the PNP transistor for the selected display. The PORTB pins activate the LED segments. For this design only the center colon decimal points were connected. The connections are RB0-dp, RB1-A, RB2-B, RB3-C...RB7-G.

The switches are also connected to PORTB I/O pins. PORTB pins RB1, RB2, and RB3 are pulled low with 10 kΩ resistors. This value is high enough to not draw current away from the LEDs when they are being driven on. Inputs are detected by pulling the pins high with a switch to V_{DD} through 820Ω resistors. This value is low enough to pull the pin high quickly when the outputs have been turned off, and to create a 90% of V_{DD} high input.

OPERATION

Switches

When no buttons are pressed, the circuit will display the current time, starting at 12:00 on reset. Pressing SW1 will cause seconds to be displayed. The time is set by pressing SW2 to advance minutes, and SW3 to advance hours. Since each of the segments are tied together across all displays, only one display should be turned on at a time, or all displays turned on would display duplicate data. The displays are turned on right to left, with each display's value being output in its turn. This is done fast enough so that there is no perceived flicker. The switches are read between display cycles.

Timing

The PIC16C5X prescaler is assigned to TMR0 as a 1:16 divide. The T0CKI pin is tied low since it is not used. The OPTION Register is loaded with 03h to initialize this prescaler setup. The software is written with timing based on a 4.000 MHz crystal. The instruction clock is 1.000 MHz after the internal divide by four. The 8-bit TMR0 register rolls over every 256 cycles, for a final frequency of 244.1406 Hz. (exactly a 4.096 ms period) A variable named sec_nth is used to count 244 roll-overs of TMR0 for one second. The benefit of keeping time with a nth variable is that it can be written to as needed to adjust time in "nths" of a second, allowing almost any odd crystal frequency to be used. Simply determine the best prescale and "nth" divider, and compute the "nth" adjustment needed for each minute, hour, twelve hour roll-over. Time can be kept accurately to two "nths" a day (an "nth" is 1/244 of a second in this case). In this circuit, 9 "nths" are subtracted each minute, 34 "nths" are added each hour, and 6 "nths" subtracted every twelve hour roll-over. This leaves a computed error of 1.5 seconds/year except for crystal frequency drift. Another possible solution is to initialize TMR0 to some value that causes a roll-over at a predetermined time interval. Writing to TMR0 causes two clock cycles to be missed while clock edges realign, which would have to be accounted for. This is described in the *Microchip Data Book*.

© 1997 Microchip Technology Inc.

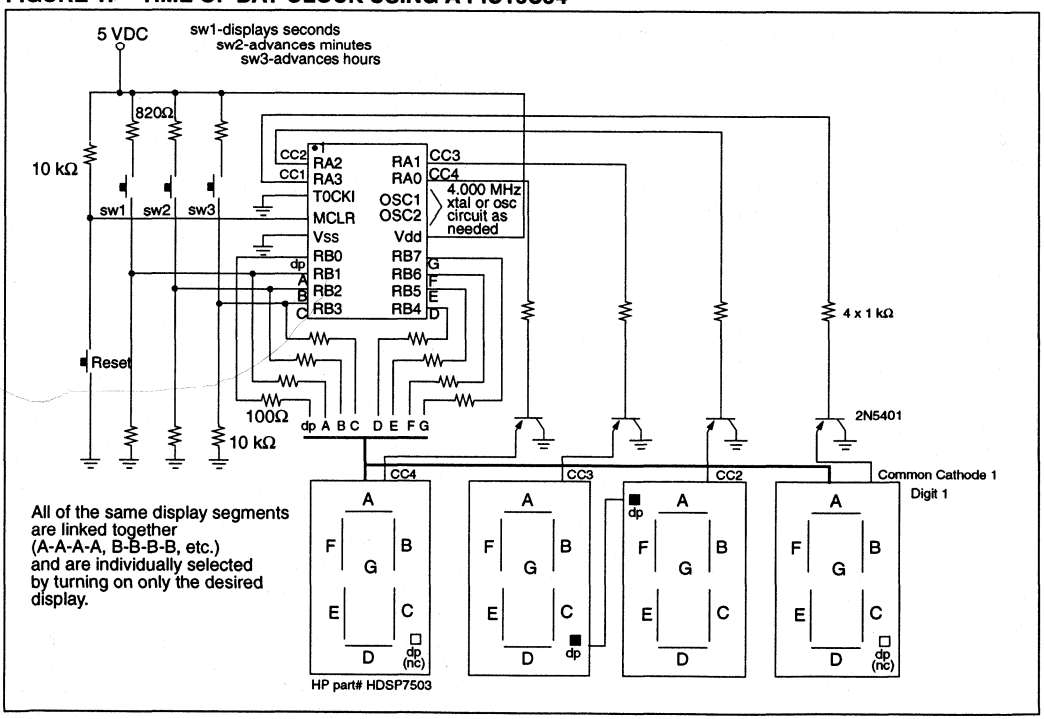
Displays

The program contains portions of code that act as a display driver. A variable exists for each of the four displays. A hex value from 0 to 9 can be written to these variables and they will be converted to display code and output to the displays. Only one display is actually on at a time, and its code is output into it in its turn. Another section of code takes the seconds, minutes, or hours value and separates it into the two digits needed for each display. In other words, 48 seconds would be separated into a "4" and an "8" and written to the appropriate display variable. The displays used were common cathode and turned on with transistors to avoid trying to sink too much current into the PIC16C5X. A display is enabled with a zero at the appropriate pin. 100Ω resistors were used in series with the segments to obtain the desired brightness. Different values may be required if different displays are used. Since each display is on less than one fourth of the time, the resistor value must be low enough to compensate for the needed forward current.

CONCLUSION

The instruction execution speed of the PIC16C54 (and the rest of the PICmicro™ series) allows many functions to be implemented with only a few pins, by multiplexing them in software. User inputs, Timer0 Counter, and multiple LED displays are all accommodated with little or no sacrifice in functionality.

FIGURE 1: TIME OF DAY CLOCK USING A PIC16C54



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com;
Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40 Released          CLOCK54.ASM   1-16-1997  17:23:58         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;
                     00002 ;*******************************************************************
                     00003         LIST    P = 16C54, n = 66
                     00004 ;
                     00005 ;                       Clock
                     00006 ;*******************************************************************
                     00007 ;
                     00008 ;                        PROGRAM DESCRIPTION
                     00009 ;
                     00010 ; This program runs on a PIC16C54.
                     00011 ;
                     00012 ;                        Hardware Description
                     00013 ;
                     00014 ;  DISPLAYS
                     00015 ; Four 7 segment displays are multiplexed.  The segments are tied together, with
                     00016 ; the common cathode pins broken out separately.  The display appears as a clock
                     00017 ; with a center semicolon ( 88:88 ).  The segments are assigned to Port B, with the
                     00018 ; semicolon being RB0, and segments A through F assigned as RB1 to RB7 respectively.
                     00019 ;  The four common cathodes are attached to the four Port A pins through transistors.
                     00020 ; RA0 for LED0, RA1/LED1... through LED3.  The center semicolon is made from the decimals
                     00021 ; of LED 2 and 3.  LED display 2 is turned upside down to put its decimal into position,
                     00022 ; but it is wired with a corrected A-F assignment to compensate.  Both decimals
                     00023 ; are tied together at RB0, but the display cathodes are still separate.
                     00024 ;
                     00025 ;  SWITCHES
                     00026 ; Because all twelve I/O pins are already used for the muxed displays, the four
                     00027 ; switches must be switched in alternatingly through software.  The switches
                     00028 ; lie across Port B pins, which wil be changed to inputs momentarily during read
                     00029 ; and changed back to outputs during display.
                     00030 ;
                     00031 ;
                     00032 ;
                     00033 ;       Program:          CLOCK54.ASM
                     00034 ;       Revision Date:
                     00035 ;                         1-16-97      Compatibility with MPASMWIN 1.40
                     00036 ;
```

© 1997 Microchip Technology Inc.

```
                    00037 ;
                    00038 ;*************************   Header ************************
                    00039 ;
                    00040 ;
000001FF            00041 PIC54    equ     H'01FF'
000003FF            00042 PIC56    equ     H'03FF'
                    00043 ;
00000000            00044 POINTER  equ     H'00'
00000001            00045 TMR0     equ     H'01'
00000002            00046 PC       equ     H'02'
00000003            00047 STATUS   equ     H'03'          ; F3 Reg is STATUS Reg.
00000004            00048 FSR      equ     H'04'
                    00049 ;
00000005            00050 PORT_A   equ     H'05'          ; 7 segment Display Common Cathodes
00000006            00051 PORT_B   equ     H'06'          ; Center Colon and Muxed Display Segments (Switches when inputs)
                    00052 ;
                    00053                                 ; STATUS REG. Bits
00000000            00054 CARRY    equ     0              ; Carry Bit is Bit.0 of F3
00000000            00055 C        equ     0
00000001            00056 DCARRY   equ     1
00000001            00057 DC       equ     1
00000002            00058 Z_bit    equ     2              ; Bit 2 of F3 is Zero Bit
00000002            00059 Z        equ     2
00000003            00060 P_DOWN   equ     3
00000003            00061 PD       equ     3
00000004            00062 T_OUT    equ     4
00000004            00063 TO       equ     4
00000005            00064 PA0      equ     5              ;16C5X Status bits
00000006            00065 PA1      equ     6              ;16C5X Status bits
00000007            00066 PA2      equ     7              ;16C5X Status bits
                    00067 ;
0000007E            00068 ZERO     equ     H'7E'
0000000C            00069 ONE      equ     H'0C'
000000B6            00070 TWO      equ     H'B6'
0000009E            00071 THREE    equ     H'9E'
000000CC            00072 FOUR     equ     H'CC'
000000DA            00073 FIVE     equ     H'DA'
000000FA            00074 SIX      equ     H'FA'          ; Mapping of segments for display (PORT_B)
0000000E            00075 SEVEN    equ     H'0E'
000000FE            00076 EIGHT    equ     H'FE'
000000CE            00077 NINE     equ     H'CE'
00000001            00078 COLON    equ     H'01'
000000F0            00079 T        equ     H'F0'
00000000            00080 BLANK    equ     H'00'
                    00081 ;
0000000C            00082 MAXNTHS  equ     D'12'          ; constants for timer variable count up
000000C4            00083 MAXSECS  equ     D'196'         ;  variables roll over in HEX at time roll over, see variable
```


© 1997 Microchip Technology Inc.

```
000000C4          00084 MAXMINS equ     D'196'          ;  explanation
000000F4          00085 MAXHRS  equ     D'244'
000000F3          00086 MINHRS  equ     D'243'
00000009          00087 ADJMIN  equ     D'9'            ; number of nths that need to be subtracted each minute
00000022          00088 ADJHR   equ     D'34'           ; nths added each hour for accurate time
00000003          00089 ADJDAY  equ     D'3'            ; nths subtracted each 1/2 day rollover
                  00090 ;
000000FE          00091 DISP1   equ     B'11111110'
000000FD          00092 DISP2   equ     B'11111101'     ; Mapping of Active Display Selection (PORT_A)
000000FB          00093 DISP3   equ     B'11111011'
000000F7          00094 DISP4   equ     B'11110111'
000000FF          00095 DISPOFF equ     H'FF'
0000000E          00096 SWITCH  equ     B'00001110'     ; Activate RB1-3 for switch inputs
                  00097 ;
                  00098 ;  Flag bit assignments
00000000          00099 SEC     equ     H'0'            ; update time display values for sec, min, or hours
00000001          00100 MIN     equ     H'1'
00000002          00101 HRS     equ     H'2'
00000003          00102 CHG     equ     H'3'            ; a change has occurred on a switch or to a potentially displayed value

00000004          00103 SW1     equ     H'4'            ; Flag bit assignments - switches that are on = 1
00000005          00104 SW2     equ     H'5'            ;  SW1 is Seconds-minutes, SW2-hours, SW3-mode
00000006          00105 SW3     equ     H'6'
00000007          00106 SW_ON   equ     H'7'            ; a switch has been pressed
                  00107 ;
                  00108 ;   VARIABLES
00000008          00109 keys    equ     H'08'           ; variable location - which keys are pressed? bit0/sw1...
00000009          00110 flags   equ     H'09'           ; bit flags; 0-SEC, 1-MIN, 2-HRS, 3-CHG, 4-SW1, 5-SW2, 6-SW3
                  00111 ;       equ     H'0A'           ; Not Used
0000000B          00112 display equ     H'0B'           ; variable location - which display to update
0000000C          00113 digit1  equ     H'0C'           ; Rightmost display value
0000000D          00114 digit2  equ     H'0D'           ; Second display from right
0000000E          00115 digit3  equ     H'0E'           ; Third     "       "     "
0000000F          00116 digit4  equ     H'0F'           ; Fourth (and Leftmost)
                  00117 ;
                  00118 ;       timer variables start at a number that allows rollover in sync with time rollover,
                  00119 ;        i.e. seconds starts at decimal 195 so that sixty 1-second increments causes 0.
00000010          00120 sec_nth equ     H'10'           ; seconds, fractional place
00000011          00121 seconds equ     H'11'           ; seconds
00000012          00122 minutes equ     H'12'           ; minutes
00000013          00123 hours   equ     H'13'           ; hours
00000014          00124 var     equ     H'14'           ; variable for misc math computations
00000015          00125 count   equ     H'15'           ; loop counter variable
00000016          00126 count2  equ     H'16'           ; 2nd loop counter for nested loops
                  00127
                  00128 ;
                  00129 ;*******************************************************************************
```

© 1997 Microchip Technology Inc.

```
                        00130 ;
                        00131 ;  Initialize Ports all outputs, blank display
                        00132 ;
0000 0C03               00133 START   movlw   H'03'           ; set option register, transition on clock,
0001 0002               00134         option                  ; Prescale TMR0, 1:16
                        00135 ;
0002 0C00               00136         movlw   0
0003 0005               00137         tris    PORT_A          ; Set all port pins as outputs
0004 0006               00138         tris    PORT_B
0005 0C00               00139         movlw   BLANK
0006 0026               00140         movwf   PORT_B          ; Blank the display
0007 04C3               00141         bcf     STATUS,PA1
0008 04A3               00142         bcf     STATUS,PA0
                        00143 ;
                        00144 ;  initialize variables
0009 0C01               00145         movlw   H'01'
000A 0021               00146         movwf   TMR0            ; set TMR0 above zero so initial wait period occurs
000B 0CFE               00147         movlw   H'FE'
000C 002B               00148         movwf   display         ; initializes display selected to first display.
000D 0C00               00149         movlw   BLANK           ; put all displays to blank, no visible segments
000E 002C               00150         movwf   digit1
000F 002D               00151         movwf   digit2
0010 002E               00152         movwf   digit3
0011 002F               00153         movwf   digit4
0012 0C0C               00154         movlw   MAXNTHS         ; set timer variables to initial values
0013 0030               00155         movwf   sec_nth
0014 0CC4               00156         movlw   MAXSECS
0015 0031               00157         movwf   seconds
0016 0CC4               00158         movlw   MAXMINS
0017 0032               00159         movwf   minutes
0018 0CFF               00160         movlw   H'FF'           ; hours start at 12 which is max at FF
0019 0033               00161         movwf   hours
001A 0C00               00162         movlw   H'00'
001B 0029               00163         movwf   flags
                        00164 ;
                        00165 ;?  call converts for minutes and hours to initialize display Variables
                        00166 ;
001C                    00167 MAIN
                        00168 ;
                        00169 ;  wait for TMR0 to roll-over
001C                    00170 TMR0_FILL
001C 0201               00171         movf    TMR0,0
001D 0743               00172         btfss   STATUS,Z        ; note, TMR0 is left free running to not lose clock cycles on writes
001E 0A1C               00173         goto    TMR0_FILL
                        00174 ;
001F 03F0               00175         incfsz  sec_nth,1       ;  add 1 to nths, n X nths = 1 sec, n is based on prescaler
0020 0A52               00176         goto    TIME_DONE
```


© 1997 Microchip Technology Inc.

```
0021 0C0C          00177          movlw   MAXNTHS
0022 0030          00178          movwf   sec_nth          ; restore sec_nths variable for next round
                   00179 ;
0023               00180 CHECK_SW
0023 07E9          00181          btfss   flags,SW_ON      ; if no switches press, bypass this
0024 0A3A          00182          goto    SET_TIME
0025 0689          00183          btfsc   flags,SW1
0026 0A3A          00184          goto    SET_TIME         ; if seconds display is pressed, do not change time
0027 0CC4          00185          movlw   MAXSECS
0028 0031          00186          movwf   seconds          ; reset seconds to zero when setting clock
0029 0C7F          00187          movlw   H'7F'
002A 0030          00188          movwf   sec_nth          ; advance second timer 1/2 second to speed time setting
002B 07A9          00189          btfss   flags,SW2
002C 0A33          00190          goto    HOURSET          ; minutes do not need changing, check hours
002D 0CAF          00191          movlw   H'AF'
002E 0030          00192          movwf   sec_nth          ; advances timer faster when setting minutes
002F 03F2          00193          incfsz  minutes,1
0030 0A33          00194          goto    HOURSET
0031 0CC4          00195          movlw   MAXMINS
0032 0032          00196          movwf   minutes
                   00197 ;
0033 06A9          00198 HOURSET  btfsc   flags,SW2
0034 0A5E          00199          goto    CHECK_TIME       ; not changing hours
0035 03F3          00200          incfsz  hours,1
0036 0A5E          00201          goto    CHECK_TIME
0037 0CF4          00202          movlw   MAXHRS
0038 0033          00203          movwf   hours
0039 0A5E          00204          goto    CHECK_TIME       ; since no timing is required, go to display changes
                   00205 ;
003A               00206 SET_TIME
003A 0509          00207          bsf     flags,SEC        ; seconds, if displayed, should be updated
003B 0569          00208          bsf     flags,CHG        ; a flag change was made.
003C 03F1          00209          incfsz  seconds,1        ;  add 1 to seconds
003D 0A52          00210          goto    TIME_DONE
003E 0CC4          00211          movlw   MAXSECS
003F 0031          00212          movwf   seconds          ; restore seconds variable for next round
                   00213 ;
0040 0529          00214          bsf     flags,MIN        ; minutes, if displayed, should be updated
0041 0569          00215          bsf     flags,CHG
0042 0C09          00216          movlw   ADJMIN
0043 00B0          00217          subwf   sec_nth,1        ; subtraction needed adjustment for each minute
0044 03F2          00218          incfsz  minutes,1        ; add 1 to minutes
0045 0A52          00219          goto    TIME_DONE
0046 0CC4          00220          movlw   MAXMINS
0047 0032          00221          movwf   minutes          ; restore minutes variable for next hour countdown
                   00222 ;
0048 0549          00223          bsf     flags,HRS
```

© 1997 Microchip Technology Inc.

```
0049 0569          00224         bsf     flags,CHG
004A 0C22          00225         movlw   ADJHR
004B 01F0          00226         addwf   sec_nth,1       ; add needed adjustment for each hour
004C 03F3          00227         incfsz  hours,1         ; add 1 to hours
004D 0A52          00228         goto    TIME_DONE
004E 0CF4          00229         movlw   MAXHRS
004F 0033          00230         movwf   hours           ; restore hours variable for next round
0050 0C03          00231         movlw   ADJDAY
0051 00B0          00232         subwf   sec_nth,1       ; subtraction adjustment for each 1/2 day rollover
                   00233 ;
0052               00234 TIME_DONE
0052 0769          00235         btfss   flags,CHG       ; if no switches or potentially dislayed numbers were
0053 0A8F          00236         goto    CYCLE           ;  changed, then skip updating display variables
                   00237 ;
                   00238 ;
0054               00239 CHECK_SECONDS
                   00240 ;  if seconds is button was pushed and not mode display seconds
0054 0789          00241         btfss   flags,SW1
0055 0A5E          00242         goto    CHECK_TIME
0056 0C00          00243         movlw   H'00'
0057 002D          00244         movwf   digit2          ; 3rd digit variable used to store temp hex value for hours display
0058 002E          00245         movwf   digit3
0059 002F          00246         movwf   digit4
005A 0CC4          00247         movlw   MAXSECS
005B 0091          00248         subwf   seconds,0
005C 002C          00249         movwf   digit1          ; 1st digit variable temporarily holds hex value for seconds display
005D 0A67          00250         goto    SPLIT_HEX
                   00251 ;
005E               00252 CHECK_TIME
005E 0C00          00253         movlw   H'00'
005F 002F          00254         movwf   digit4          ; zero out tens places in case there is no tens increment
0060 002D          00255         movwf   digit2
0061 0CF3          00256         movlw   MINHRS
0062 0093          00257         subwf   hours,0
0063 002E          00258         movwf   digit3          ; 3rd digit variable temporarily holds hex value for hours
0064 0CC4          00259         movlw   MAXMINS
0065 0092          00260         subwf   minutes,0
0066 002C          00261         movwf   digit1          ; 1st digit temporarily holds hex value for minutes
                   00262 ;
                   00263 ;
                   00264 ;
0067               00265 SPLIT_HEX                       ;  split into two hex display variables and write
                   00266 ;
0067 0C02          00267         movlw   H'02'
0068 0035          00268         movwf   count           ; loop to convert each number - seconds - or minutes and hours
                   00269
                   00270 ;1st time through, FSR = digit1, 2nd time FSR = digit3
```


© 1997 Microchip Technology Inc.

```
0069 0C0C          00271          movlw   digit1          ;
006A 0024          00272          movwf   FSR            ; address of digit1 into File Select Register enables POINTER
006B 0A6E          00273          goto    LOOP           ; this loop is used to modify the minutes/seconds place
                   00274 ;
006C 0C0E          00275 LOOP2    movlw   digit3
006D 0024          00276          movwf   FSR            ; this loop is used to modify the hours place
                   00277 ;
006E               00278 LOOP
006E 0C0A          00279          movlw   D'10'
006F 00A0          00280          subwf   POINTER,1      ; find out how many tens in number,
0070 0603          00281          btfsc   STATUS,C       ; was a borrow needed?
0071 0A74          00282          goto    INCREMENT_10S  ;  if not, add 1 to tens position
0072 01E0          00283          addwf   POINTER,1      ;  if so, do not increment tens place, add ten back on to get 1s
0073 0A78          00284          goto    NEXT_DIGIT
                   00285 ;
0074               00286 INCREMENT_10S
0074 02A4          00287          incf    FSR,1          ; bump address pointed to from 1st position to 10s
0075 02A0          00288          incf    POINTER,1      ; add 1 to 10s position as determined by previous subtract
0076 00E4          00289          decf    FSR,1          ; put POINTER value back to 1s place for next subtraction
0077 0A6E          00290          goto    LOOP           ; go back and keep subtracting until finished
                   00291 ;
0078               00292 NEXT_DIGIT
0078 02F5          00293          decfsz  count,1
0079 0A6C          00294          goto    LOOP2
                   00295 ;
007A               00296 CONVERT_HEX_TO_DISPLAY          ; converts hex number in digit variables to decimal display code
007A 0C0C          00297          movlw   digit1
007B 0024          00298          movwf   FSR            ; put the address of the first digit into the FSR to enable POINTER
007C 0C04          00299          movlw   H'04'
007D 0035          00300          movwf   count          ; prepare count variable to loop for all four displays
007E               00301 NEXT_HEX
007E 0200          00302          movf    POINTER,0      ; get the hex value of the current digit variable
007F 09C0          00303          call    RETURN_CODE    ; call for the hex to decimal display conversion
0080 0020          00304          movwf   POINTER        ; put the returned display code back into the digit variable
0081 02A4          00305          incf    FSR,1          ; increment the pointer to the next digit variable address
0082 02F5          00306          decfsz  count,1        ; allow only count(4) times through loop
0083 0A7E          00307          goto    NEXT_HEX
                   00308 ;
0084               00309 FIX_DISPLAY
0084 0C7E          00310          movlw   ZERO
0085 008F          00311          subwf   digit4,0
0086 0743          00312          btfss   STATUS,Z
0087 0A8A          00313          goto    FIX_SEC
0088 0C00          00314          movlw   BLANK
0089 002F          00315          movwf   digit4
                   00316
008A 0789          00317 FIX_SEC  btfss   flags,SW1
```

© 1997 Microchip Technology Inc.

```
008B 0A8D          00318          goto    CLEAR_FLAGS
008C 002E          00319          movwf   digit3
                   00320 ;
008D               00321 CLEAR_FLAGS
008D 0CF0          00322          movlw   H'F0'
008E 0169          00323          andwf   flags,1         ; clear the lower 4 flag bits to show update status
                   00324 ;
008F               00325 CYCLE
                   00326 ;
008F 0CFF          00327          movlw   DISPOFF
0090 0025          00328          movwf   PORT_A          ; Turn off LED Displays
0091 0C0E          00329          movlw   SWITCH
0092 0006          00330          tris    PORT_B          ; Set some port B pins as switch inputs
0093 0C0F          00331          movlw   H'0F'
0094 0169          00332          andwf   flags,1         ; reset switch flags to zero
0095 0000          00333          nop                     ; nop may not be needed, allows old outputs to bleed
0096 0000          00334          nop                     ;   off through 10k R before reading port pins
0097 0000          00335          nop
0098 0206          00336          movf    PORT_B,0
0099 0034          00337          movwf   var
009A 0734          00338          btfss   var,1
009B 0A9F          00339          goto    SWITCH2
009C 0569          00340          bsf     flags,CHG
009D 0589          00341          bsf     flags,SW1
009E 05E9          00342          bsf     flags,SW_ON
009F 0754          00343 SWITCH2  btfss   var,2
00A0 0AA4          00344          goto    SWITCH3
00A1 0569          00345          bsf     flags,CHG
00A2 05A9          00346          bsf     flags,SW2
00A3 05E9          00347          bsf     flags,SW_ON
00A4 0774          00348 SWITCH3  btfss   var,3
00A5 0AA9          00349          goto    SETPORT
00A6 0569          00350          bsf     flags,CHG
00A7 05C9          00351          bsf     flags,SW3
00A8 05E9          00352          bsf     flags,SW_ON
                   00353 ;
00A9 0C00          00354 SETPORT  movlw   H'00'
00AA 0006          00355          tris    PORT_B
00AB 0C00          00356          movlw   BLANK
00AC 0026          00357          movwf   PORT_B
                   00358 ;
                   00359 ;   determine which display needs updating and cycle it on
00AD 070B          00360          btfss   display,0       ; if 1st display, get 1st digit
00AE 020F          00361          movf    digit4,0
00AF 072B          00362          btfss   display,1       ; if 2nd display, get 2nd digit
00B0 020E          00363          movf    digit3,0
00B1 074B          00364          btfss   display,2       ; if 3rd display, get 3rd digit
```


© 1997 Microchip Technology Inc.

```
00B2 020D           00365         movf    digit2,0
00B3 076B           00366         btfss   display,3      ; if 4th display, get 4th digit
00B4 020C           00367         movf    digit1,0
00B5 0026           00368         movwf   PORT_B         ; put the number out to display
00B6 06F0           00369         btfsc   sec_nth,7
00B7 0506           00370         bsf     PORT_B,0       ; sets colon decimal on %50 duty using highest bit
00B8 020B           00371         movf    display,0      ; get display needing cycle on
00B9 0025           00372         movwf   PORT_A         ; enables proper display
00BA 002B           00373         movwf   display        ; returns old w if not done, new w if resetting display
00BB 036B           00374         rlf     display,1      ; rotate display "on" bit to next position
00BC 050B           00375         bsf     display,0      ; assures a 1 on lowest position since rotated carry is zero
00BD 078B           00376         btfss   display,4      ; check if last display was already updated
00BE 040B           00377         bcf     display,0      ; if it was, set display back to 1st (bit 0 set)
                    00378 ;
                    00379 ;
                    00380 ;
00BF 0A1C           00381         goto    MAIN
                    00382 ;
00C0                00383 RETURN_CODE
                    00384 ;
00C0 01E2           00385         addwf   PC,1
00C1 087E           00386         retlw   ZERO
00C2 080C           00387         retlw   ONE
00C3 08B6           00388         retlw   TWO
00C4 089E           00389         retlw   THREE
00C5 08CC           00390         retlw   FOUR
00C6 08DA           00391         retlw   FIVE
00C7 08FA           00392         retlw   SIX
00C8 080E           00393         retlw   SEVEN
00C9 08FE           00394         retlw   EIGHT
00CA 08CE           00395         retlw   NINE
                    00396 ;
                    00397 ;
01FF                00398         org     PIC54
01FF 0A00           00399         goto    START
                    00400 ;
                    00401     END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXX----- ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.
```

© 1997 Microchip Technology Inc.

```
Program Memory Words Used:   204
Program Memory Words Free:   308


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Frequency Counter Using PIC16C5X

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

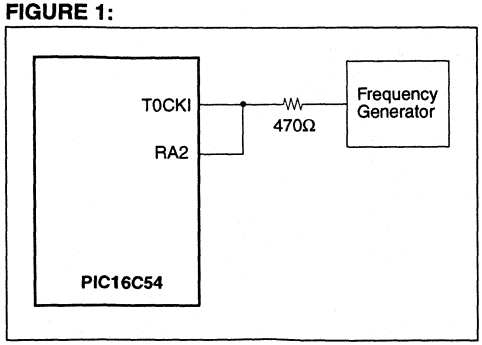
The PIC16C5X has one 8-bit timer (Timer0), which can be used with an 8-bit prescaler. The prescaler runs asynchronously, hence it can count a very high frequency. The minimum rise and fall times of the input frequency are specified to be 10 ns, so the fastest clock rate the TMR0 can count is 50 MHz. The prescaler must be used when measuring high frequency. Since the prescaler can be configured as a divide by 256 counter, the maximum resolution at which the input frequency can be measured is 16-bits. However, the prescaler cannot be directly read like a file register. This application note depicts a unique method by which the user can "extract" the 8-bit value in the prescaler, whereby the resolution of the measurement is 16-bits with the high 8-bits in TMR0 and the low 8-bits in the prescaler.

IMPLEMENTATION

A frequency counter which can read frequencies from 50 Hz to 50 MHz was implemented in this application note in order to demonstrate this method of measuring the 16-bit counter value from the prescaler and TMR0.

The basic hardware for the measurement circuit is depicted in Figure 1. It consists of the frequency input at TMR0 or T0CKI (pin 3 in a PIC16C54). T0CKI is connected to RA2. The input frequency is connected to T0CKI through a 470Ω resistor.

FIGURE 1:



TMR0 is configured to measure the input frequency, at T0CKI of the PIC16C54. The input frequency is "gated" for a precise duration of time. Before starting this precise "gate", TMR0 is cleared (which also clears the prescaler), and the RA2 pin is configured as an input. The precise "gate" is implemented in software as an accurate delay. At the end of the delay, the RA2 pin is configured as an output going low. This will cause the input to TMR0 to be "halted" or "stopped". A 16-bit value of the input frequency is now saved in TMR0 and the 8-bit prescaler. The high 8 bits are in TMR0 and can be easily read. The low 8 bits have to be "shifted out". The 8 bits in the prescaler are "shifted out" by toggling RA2 with a "BSF" and a "BCF" instruction. After every toggle, the value in TMR0 is checked to see if TMR0 has incremented. If the number of toggles required to cause TMR0 to increment by 1 is N, then the 8-bit value in the prescaler can be calculated to be = (256 - N). By concatenating the calculated value and the original value from TMR0, the 16-bit value for the frequency is determined.

To measure a wide range of frequencies, the following intermediate steps were taken:

Frequency Range	Precise "gate" delay	Resolution
50 MHz - 10 MHz	1 ms	±10 kHz
10 MHz - 1 MHz	5 ms	±2 kHz
1 MHz - 100 kHz	50 ms	±200 Hz
100 Hz - 10 kHz	200 ms	±50 Hz
50 Hz - 50 Hz	50 ms (†)	±2 Hz

Note: In this case, TMR0 uses the internal 4 MHz clock and counts the number of instances of the external clock. The maximum time required is 50 ms to make a ± 2 Hz accurate measurement for 10 kHz input frequency.

The check for the correct frequency is performed automatically starting with the high frequency and ending with the low frequency. The maximum time required for each conversion is approximately 310 ms. In other words, three frequency checks are done every second.

CONCLUSION

The PIC16C5X family can be used to make a 16-bit measurement of input frequency with a small overhead of one resistor and one I/O port.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40 Released              FREQ.ASM   1-16-1997  17:29:41         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         list p=16C54
                     00002 ;
                     00003         include "p16c5x.inc"
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File, Version 3.30 Microchip Technology, Inc.
                     00224         LIST
                     00004
                     00005 #define         _ra0            PORTA,0
                     00006 #define         _ra1            PORTA,1
                     00007
                     00008 ;
                     00009 ;This program implements the concepts for the frequency counter
                     00010 ;using a PIC16C54. In this program, RA0 is connected directly
                     00011 ;to the tmr0 input. Tmr0 input is connected thru a 470 ohm
                     00012 ;resistor to the freq source. Please note that the
                     00013 ;the input freq. is required to be a 50% duty cycle, square
                     00014 ;wave. Though none of the internal calculations are based
                     00015 ;on this requirement, waveforms which deviate drastically
                     00016 ;for the one specified were not tested using these routines.
                     00017 ;The routines written in this program, automatically measure
                     00018 ;waveforms from 50MHz to 50hz in a period of approx. 300 mS.
                     00019 ;After a period of approx 300 mS, the 16 bit "measured" value of
                     00020 ;the freq. is read and saved in the location "flo" and "fhi".
                     00021 ;A "range" flag is set to indicate if the measurement belongs to
                     00022 ;the five ranges measured namely:
                     00023 ;       RANGE:                  Flag name
                     00024 ;       50Mhz to 10Mhz -->  Mhz 50 to 10
                     00025 ;       10Mhz to 1Mhz  -->  Mhz 10 to 1
                     00026 ;       1Mhz to 100Khz -->  Khz 1K to 100
                     00027 ;       100Khz to 10Khz --> Khz 100 to 10
                     00028 ;       10Khz to 50hz  -->  Hz 10K to 50
                     00029 ;The freq. check is repeated to give approx 3 samples/sec.
                     00030 ;The "measured" value now has to go through a calculation to
                     00031 ;get the actual value. Please use the math routines mentioned
                     00032 ;elsewhere in the Embedded Control Handbook to determine
                     00033 ;the actual value of the freq.
                     00034 ;*****************************************************************
                     00035 ;Calculations required to determine actual freq. values
                     00036 ;*****************************************************************
                     00037 ;First determine which range flag is set, then calculate as follows:
                     00038 ;
                     00039 ;       Mhz50to10:  freq. = (fhi|flo) X 1000
                     00040 ;       Mhz10to1:   freq. = (fhi|flo) X 200
                     00041 ;       Khz1Kto100: freq. = (fhi|flo) X 20
                     00042 ;       Khz100to10: freq. = (fhi|flo) X 5
                     00043 ;       Hz10Kto50: Please see comments above routine Freq10Kto50
                     00044 ;
                     00045 ;
                     00046 ;       Program:          FREQ.ASM
                     00047 ;       Revision Date:
                     00048 ;                         1-16-97      Compatibility with MPASMWIN 1.40
                     00049 ;
                     00050 ;*****************************************************************
                     00051 ;
```


© 1997 Microchip Technology Inc.

```
0000000B   00052 fhi      equ             .11             ;high 8 bit value for freq.
0000000A   00053 flo      equ             .10             ;low 8 bit value for freq.
0000000C   00054 tempa    equ             .12
0000000D   00055 tempb    equ             .13
0000000D   00056 limithi  equ             .13
0000000C   00057 limitlo  equ             .12
0000000D   00058 count    equ             .13
0000000E   00059 trisabuf equ             .14
00000010   00060 InputCounthi             equ     .16
0000000F   00061 InputCountlo             equ     .15
           00062 #define ddra0          trisabuf,0
00000011   00063 RangeFlag               equ     .17
           00064 #define Mhz50to10      RangeFlag,0
           00065 #define Mhz10to1       RangeFlag,1
           00066 #define Khz1Kto100     RangeFlag,2
           00067 #define Khz100to10     RangeFlag,3
           00068 #define Hz10Kto50      RangeFlag,4
           00069 #define RangeError     RangeFlag,5
           00070 ;
00002710   00071 tenMhz   equ             .10000000/.1000
00001388   00072 oneMhz   equ             .1000000/.200
00001388   00073 hndredK  equ             .100000/.20
000007D0   00074 tenKhz   equ             .10000/.5
           00075 ;
00000001   00076 Debug    equ             1
           00077 ;
           00078 enabletmr0               macro
           00079         clrf            TMR0
           00080         bsf             ddra0
           00081         movf            trisabuf,W
           00082         tris            PORTA
           00083         endm
           00084 ;
           00085 disabletmr0              macro
           00086         bcf             ddra0
           00087         bcf             _ra0
           00088         movf            trisabuf,W
           00089         tris            PORTA
           00090         endm
           00091 ;
01FF       00092         org     0x1ff
01FF 0A00  00093         goto            start
0000       00094         org     0
0000       00095 start
0000 0C0F  00096         movlw   0x0f                    ;initialize ddra
0001 002E  00097         movwf           trisabuf        ;       /
           00098         disabletmr0
0002 040E             M         bcf             ddra0
0003 0405             M         bcf             _ra0
0004 020E             M         movf            trisabuf,W
0005 0005             M         tris            PORTA
0006 0C37  00099         movlw           B'00110111'     ;set the option register
0007 0002  00100         option                          ;to measure high freq.
0008 0066  00101         clrf            PORTB
0009 0040  00102         clrw
000A 0006  00103         tris            PORTB
           00104
000B       00105 repeat
           00106         enabletmr0                      ;enable tmr0
000B 0061             M         clrf            TMR0
000C 050E             M         bsf             ddra0
000D 020E             M         movf            trisabuf,W
000E 0005             M         tris            PORTA
000F 09BA  00107                 call            delay1mS ;wait for 1mS
           00108         disabletmr0                     ;disable tmr0
0010 040E             M         bcf             ddra0
```

© 1997 Microchip Technology Inc.

```
0011 0405              M        bcf          _ra0
0012 020E              M        movf         trisabuf,W
0013 0005              M        tris         PORTA
0014 09E1   00109        call          getfreq          ;get freq in fhi and flo
0015 097C   00110        call          check10M         ;check if <= 10 Mhz
0016 0743   00111        btfss         STATUS,Z         ;yes then do lower freq.
0017 0A9F   00112        goto          Freq50Mto10M     ;found 50Mhz to 10Mhz freq.
            00113        enabletmr0                     ;enable tmr0
0018 0061              M        clrf         TMR0
0019 050E              M        bsf          ddra0
001A 020E              M        movf         trisabuf,W
001B 0005              M        tris         PORTA
001C 09C3   00114        call          delay5mS         ;wait for 5mS
            00115        disabletmr0                    ;disable tmr0
001D 040E              M        bcf          ddra0
001E 0405              M        bcf          _ra0
001F 020E              M        movf         trisabuf,W
0020 0005              M        tris         PORTA
0021 09E1   00116        call          getfreq          ;get freq in fhi and flo
0022 0990   00117        call          check1M          ;check if <= 1 Mhz
0023 0743   00118        btfss         STATUS,Z         ;yes then do lower freq.
0024 0AA2   00119        goto          Freq10Mto1M      ;else wait for 300 mS
            00120        enabletmr0                     ;enable tmr0
0025 0061              M        clrf         TMR0
0026 050E              M        bsf          ddra0
0027 020E              M        movf         trisabuf,W
0028 0005              M        tris         PORTA
0029 09CD   00121        call          delay50mS        ;wait for 50mS
            00122        disabletmr0                    ;disable tmr0
002A 040E              M        bcf          ddra0
002B 0405              M        bcf          _ra0
002C 020E              M        movf         trisabuf,W
002D 0005              M        tris         PORTA
002E 09E1   00123        call          getfreq          ;get freq in fhi and flo
002F 0995   00124        call          check100K        ;check if <= 100 Khz
0030 0743   00125        btfss         STATUS,Z         ;yes then do lower freq.
0031 0AA5   00126        goto          Freq1Mto100K     ;else wait for 250 mS
            00127        enabletmr0                     ;enable tmr0
0032 0061              M        clrf         TMR0
0033 050E              M        bsf          ddra0
0034 020E              M        movf         trisabuf,W
0035 0005              M        tris         PORTA
0036 09D7   00128        call          delay200mS       ;wait for 200 mS
            00129        disabletmr0                    ;disable tmr0
0037 040E              M        bcf          ddra0
0038 0405              M        bcf          _ra0
0039 020E              M        movf         trisabuf,W
003A 0005              M        tris         PORTA
003B 09E1   00130        call          getfreq          ;get freq in fhi and flo
003C 099A   00131        call          check10K         ;check if <= 10Khz
003D 0743   00132        btfss         STATUS,Z         ;yes then do lower freq.
003E 0AA8   00133        goto          Freq100Kto10K    ;else wait 50mS
            00134 ;
            00135 ;*****************************************************************
            00136 ;The freq. below 10khz to 50hz is got by using the input freq.
            00137 ;to gate the internal 4Mhz clock. The gate is not  "opened"
            00138 ;until a leading or falling transition is observed at the input.
            00139 ;For approx. 50 mS, the internal 1uS clock is sourced to
            00140 ;the TMR0 with a divide by 256 prescaler.  Every 20uS or so,
            00141 ;the transitions on the input line are checked. If a transition
            00142 ;is observed, then the "InputCount" is incremented. At the end of 50mS,
            00143 ;a last transition is used to close the gate and stop the measurement
            00144 ;of the internal freq.
            00145 ;Say the input freq to be measured is 1500hz. In 50mS, approx  75
            00146 ;cycles will be counted in InputCount. The 16 bit value in flo
            00147 ;and fhi is approx. 50,000. Then the freq measured:
```


© 1997 Microchip Technology Inc.

```
                00148 ;
                00149 ;                 freq. = 75 X 1,000,000/60,000 = 1500 in this case
                00150 ; In general      freq. = InputCount X 1,000,000/(fhi|flo).
                00151 ;
003F            00152 Freq10Kto50
003F 0070       00153           clrf            InputCounthi    ;0 --> InputCount
0040 006F       00154           clrf            InputCountlo    ;       /
0041 0C17       00155           movlw           B'00010111'     ;start TMR0 with internal
0042 0002       00156           option                          ; clk. = 1uS
0043 0C0F       00157           movlw           B'00001111'     ;set RA0 as a input
0044 0005       00158           tris            PORTA           ;       /
0045 0705       00159           btfss           _ra0            ;see if level low
0046 0A49       00160           goto            FirstHigh       ;yes then check leading edge
0047            00161 FirstLow
0047 0605       00162           btfsc           _ra0            ;else look for falling edge
0048 0A47       00163           goto            FirstLow        ;               /
0049            00164 FirstHigh                                 ;and look for first high
0049 0705       00165           btfss           _ra0            ;look for first high
004A 0A49       00166           goto            FirstHigh       ;       /
004B 0061       00167           clrf            TMR0            ;start count
004C 0CC3       00168           movlw           high .50000     ;get high byte of 50000
004D 002D       00169           movwf           limithi         ;save in RAM
004E            00170 NextLow
004E 0201       00171           movf            TMR0,W          ;50mS over?
004F 008D       00172           subwf           limithi,W       ;approx. 50
0050 0643       00173           btfsc           STATUS,Z        ;no then skip
0051 0A65       00174           goto            LastHigh        ;look for lasthigh
0052 0605       00175           btfsc           _ra0            ;look for low
0053 0A4E       00176           goto            NextLow         ;   /
0054            00177 NextHigh
0054 0201       00178           movf            TMR0,W          ;50mS over?
0055 008D       00179           subwf           limithi,W       ;approx. 50
0056 0643       00180           btfsc           STATUS,Z        ;no then skip
0057 0A5E       00181           goto            LastLow         ;look for lastlow
0058 0705       00182           btfss           _ra0
0059 0A54       00183           goto            NextHigh
005A 02AF       00184           incf            InputCountlo, F ;inc count
005B 0643       00185           btfsc           STATUS,Z        ;overflow?
005C 02B0       00186           incf            InputCounthi, F ;inc high value
005D 0A4E       00187           goto            NextLow         ;check next
005E            00188 LastLow
005E 0201       00189           movf            TMR0,W          ;tmr0 overflow?
005F 002C       00190           movwf           tempa           ;       /
0060 02AC       00191           incf            tempa, F        ;       /
0061 0643       00192           btfsc           STATUS,Z                ;no then skip
0062 0A6C       00193           goto            CloseGate       ;overflow then abort
0063 0605       00194           btfsc           _ra0            ;look for low
0064 0A5E       00195           goto            LastLow         ;       /
0065            00196 LastHigh
0065 0201       00197           movf            TMR0,W         ;tmr0 overflow?
0066 002C       00198           movwf           tempa           ;       /
0067 02AC       00199           incf            tempa, F        ;       /
0068 0643       00200           btfsc           STATUS,Z        ;no then skip
0069 0A6C       00201           goto            CloseGate       ;overflow then abort
006A 0705       00202           btfss           _ra0            ;look for high
006B 0A65       00203           goto            LastHigh
006C            00204 CloseGate
006C 0C27       00205           movlw           B'00100111'     ;stop internal clk
006D 0002       00206           option                          ;       /
                00207           disabletmr0                     ;disable tmr0
006E 040E                 M         bcf             ddra0
006F 0405                 M         bcf             _ra0
0070 020E                 M         movf            trisabuf,W
0071 0005                 M         tris            PORTA
0072 09E1       00208           call            getfreq         ;get freq
0073 028B       00209           incf            fhi,W           ;out of range?
```

```
0074 0643    00210          btfsc          STATUS,Z        ;       /
0075 0A79    00211          goto           OutofRange      ;yes then set flag
0076 0071    00212          clrf           RangeFlag       ;set Hz10Kto50 flag
0077 0591    00213          bsf            Hz10Kto50       ;       /
0078 0AAB    00214          goto           wait50mS
0079         00215 OutofRange
0079 0071    00216          clrf           RangeFlag       ;set error flag
007A 05B1    00217          bsf            RangeError
007B 0AAB    00218          goto           wait50mS
             00219 ;
             00220 ;Check10M, check if the freq < 10 Mhz if yes then the z bit
             00221 ;is set else it is cleared. This routine uses a generic routine
             00222 ;checklimit, which check the value in fhi and flo to the ones
             00223 ;in limithi and limitlo
007C         00224 check10M
007C 0C27    00225          movlw          high tenMhz     ;get hi value of 10Mhz
007D 002D    00226          movwf          limithi         ;save in limithi
007E 0C10    00227          movlw          low tenMhz      ;get lo value of 10Mhz
007F 002C    00228          movwf          limitlo         ;save in limitlo
             00229 ;checklimit, checks if the freq in flo and fhi is lower
             00230 ;than the values set in limitlo and limithi. It is a
             00231 ;common routine used to check all set limits. If the value
             00232 ;is <= the z bit = 0 else z = 1 .
0080         00233 checklimit
0080 020B    00234          movf           fhi,W           ;get high freq value
0081 00AD    00235          subwf          limithi, F      ;and check with high value
0082 0643    00236          btfsc          STATUS,Z        ;if not equal then skip
0083 0A88    00237          goto           chk10Mlo        ;else check low value
0084 0703    00238          btfss          STATUS,C        ;skip if value is < limit
0085 0800    00239          retlw    0                     ;value > limit so z = 0.
0086 0040    00240          clrw                           ;z = 1
0087 0800    00241          retlw    0                     ;return with z flag set
0088         00242 chk10Mlo
0088 020A    00243          movf           flo,W           ;get low value
0089 00AC    00244          subwf          limitlo, F      ;and check with low value
008A 0643    00245          btfsc          STATUS,Z        ;not equal then skip
008B 0800    00246          retlw    0                     ;else return with z = 1
008C 0703    00247          btfss          STATUS,C        ;skip if value is < limit
008D 0800    00248          retlw    0                     ;value > limit so z = 0
008E 0040    00249          clrw                           ; z = 1
008F 0800    00250          retlw    0                     ;return with z flag set
             00251 ;
             00252 ;Check1M checks if freq is below 1 Mhz
             00253 ;
0090         00254 check1M
0090 0C13    00255          movlw          high oneMhz     ;get hi value of 1Mhz
0091 002D    00256          movwf          limithi         ;save in limithi
0092 0C88    00257          movlw          low oneMhz      ;get lo value of 1Mhz
0093 002C    00258          movwf          limitlo         ;save in limitlo
0094 0A80    00259          goto           checklimit
             00260 ;
0095         00261 check100K
0095 0C13    00262          movlw          high hndredK    ;get hi value of 100Khz
0096 002D    00263          movwf          limithi         ;save in limithi
0097 0C88    00264          movlw          low hndredK     ;get lo value of 100Khz
0098 002C    00265          movwf          limitlo         ;save in limitlo
0099 0A80    00266          goto           checklimit
             00267 ;
009A         00268 check10K
009A 0C07    00269          movlw          high tenKhz     ;get hi value of 10Khz
009B 002D    00270          movwf          limithi         ;save in limithi
009C 0CD0    00271          movlw          low tenKhz      ;get lo value of 10Khz
009D 002C    00272          movwf          limitlo         ;save in limitlo
009E 0A80    00273          goto           checklimit
             00274 ;
             00275 ;
```


© 1997 Microchip Technology Inc.

```
009F            00276 Freq50Mto10M
009F 0071       00277         clrf            RangeFlag
00A0 0511       00278         bsf             Mhz50to10
00A1 0AAB       00279         goto            wait300mS
00A2            00280 Freq10Mto1M
00A2 0071       00281         clrf            RangeFlag
00A3 0531       00282         bsf             Mhz10to1
00A4 0AAB       00283         goto            wait300mS
00A5            00284 Freq1Mto100K
00A5 0071       00285         clrf            RangeFlag
00A6 0551       00286         bsf             Khz1Kto100
00A7 0AAB       00287         goto            wait250mS
00A8            00288 Freq100Kto10K
00A8 0071       00289         clrf            RangeFlag
00A9 0571       00290         bsf             Khz100to10
00AA 0AAB       00291         goto            wait50mS
                00292 ;
00AB            00293 wait300mS
                00294     If      !Debug
                00295         call            delay50mS
                00296     ENDIF
00AB            00297 wait250mS
                00298     IF      !Debug
                00299         call            delay50mS
                00300         call            delay50mS
                00301         call            delay50mS
                00302         call            delay50mS
                00303     ENDIF
00AB            00304 wait50mS
                00305     IF      !Debug
                00306         call            delay50mS
                00307     ENDIF
                00308 ;
                00309 ;
                00310     IF      Debug
                00311 ;This routine debugs freq. on a PICDEM1 board.
00AB            00312 checkRA1
00AB 0625       00313         btfsc           _ra1
00AC 0AAB       00314         goto            checkRA1
00AD 09D7       00315         call            delay200mS
00AE 020B       00316         movf            fhi,W
00AF 0026       00317         movwf           PORTB
00B0            00318 chkRA1hi
00B0 0725       00319         btfss           _ra1
00B1 0AB0       00320         goto            chkRA1hi
00B2            00321 chkRA1lo
00B2 0625       00322         btfsc           _ra1
00B3 0AB2       00323         goto            chkRA1lo
00B4 09D7       00324         call            delay200mS
00B5 020A       00325         movf            flo,W
00B6 0026       00326         movwf           PORTB
00B7 0725       00327         btfss           _ra1
00B8 0AB7       00328         goto            $-1
                00329     ENDIF
00B9 0A0B       00330         goto            repeat
                00331 ;
                00332 ;delay1ms, is a very accurate 1mS delay for a 4Mhz clock.
00BA            00333 delay1mS
00BA 0CC5       00334         movlw           .197
00BB 002D       00335         movwf           count
00BC 0000       00336         nop
00BD 0ABE       00337         goto            $+1
00BE 0ABF       00338         goto            $+1
00BF            00339 dly1mS
00BF 0AC0       00340         goto            $+1
00C0 02ED       00341         decfsz          count, F
```


© 1997 Microchip Technology Inc.

```
00C1 0ABF     00342          goto            dly1mS
00C2 0800     00343          retlw    0
              00344 ;
              00345 ;delay5mS uses delay1mS to get a very accurate 5 mS delay
00C3          00346 delay5mS
00C3 09BA     00347          call            delay1mS
00C4 09BA     00348          call            delay1mS
00C5 09BA     00349          call            delay1mS
00C6 09BA     00350          call            delay1mS
00C7 09BA     00351          call            delay1mS
00C8 0C04     00352          movlw           .4
00C9 002D     00353          movwf           count
00CA          00354 tweek5mS
00CA 02ED     00355          decfsz          count, F
00CB 0ACA     00356          goto            tweek5mS
00CC 0800     00357          return
              00358 ;
              00359 ;delay50mS uses delay1mS to get a very accurate 50mS delay
00CD          00360 delay50mS
00CD 0C32     00361          movlw           .50
00CE 002C     00362          movwf           tempa
00CF          00363 dly50mS
00CF 09BA     00364          call            delay1mS
00D0 02EC     00365          decfsz          tempa, F
00D1 0ACF     00366          goto            dly50mS
00D2 0C0E     00367          movlw           .14
00D3 002D     00368          movwf           count
00D4          00369 tweek50mS
00D4 02ED     00370          decfsz          count, F
00D5 0AD4     00371          goto            tweek50mS
00D6 0800     00372          retlw    0
              00373 ;
              00374 ;delay200mS uses delay1mS to get a very accurate 200mS delay.
00D7          00375 delay200mS
00D7 0CC8     00376          movlw           .200
00D8 002C     00377          movwf           tempa
00D9          00378 dly200mS
00D9 09BA     00379          call            delay1mS
00DA 02EC     00380          decfsz          tempa, F
00DB 0AD9     00381          goto            dly200mS
00DC 0C40     00382          movlw           .64
00DD 002D     00383          movwf           count
00DE          00384 tweek200mS
00DE 02ED     00385          decfsz          count, F
00DF 0ADE     00386          goto            tweek200mS
00E0 0800     00387          retlw    0
              00388 ;
              00389 ;getfreq, toggles the RA0 pin  to shift out the value in the
              00390 ;prescaler. The number of toggles is kept in count. If the value
              00391 ;in tmr0 increments, then the low 8 bit value = !count + 1. The low
              00392 ;value of the freq. is loaded in flo and the high in fhi.
00E1          00393 getfreq
00E1 0201     00394          movf            TMR0,W          ;get the tmr0 value
00E2 002B     00395          movwf           fhi             ;save in fhi
00E3 006D     00396          clrf            count           ;keep track of the toggles
00E4          00397 toggle
00E4 02AD     00398          incf            count, F        ;inc for first
00E5 0405     00399          bcf             _ra0            ;toggle the input
00E6 0505     00400          bsf             _ra0            ;       /
00E7 0201     00401          movf            TMR0,W          ;see if tmr0 incremented
00E8 008B     00402          subwf           fhi,W           ;       /
00E9 0643     00403          btfsc           STATUS,Z        ;yes then skip
00EA 0AE4     00404          goto            toggle          ;no then toggle again
00EB 026D     00405          comf            count, F        ;else complement count
00EC 028D     00406          incf            count,W         ;and increment
00ED 002A     00407          movwf                   flo     ;save in flo
```


© 1997 Microchip Technology Inc.

```
00EE 0800     00408          retlw          0                      ;return
              00409 ;
              00410          end

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)


0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   240
Program Memory Words Free:   272


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Serial Port Routines Without Using Timer0

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

The PIC16C5X has one 8-bit timer (Timer0) which can use an 8-bit prescaler. In some instances, the user would like to use this timer for some other purpose and yet, be able to do a transmit and receive using the serial port. This application note offers routines to do a simple 8-bit transmit and receive with no handshake, at baud rates from 1200 to 9600. Please note that these routines use a timed loop which is as accurate as the clock which drives the PIC16C5X. The user enters the frequency and baud rate desired. The calculated value "delay" in the serial routine has to be an 8-bit value only. If the value is greater than 8-bits, the frequency and baud rate values have to be changed.

CONCLUSION

Simple transmit and receive routines can be written without using Timer0 to generate the baud rate.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40 Released           SER54.ASM   1-16-1997  17:32:17         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;
                     00002 ;These routines were written to work on the PICDEM1 hardware.
                     00003 ;The frequency of the clock is 16Mhz and the hardware uses no
                     00004 ;handshake
                     00005 ;       TX -> RA3
                     00006 ;       RX -> RA2
                     00007 ;
                     00008 ;   Program:          SER54.ASM
                     00009 ;   Revision Date:
                     00010 ;                  12-12-95  Compatibility with MPASMWIN 1.30
                     00011 ;
                     00012 ;***************************************************************************
                     00013 ;
                     00014         list    p=16c54
                     00015 ;
  00F42400           00016 clockrate equ   .16000000       ;define clock rate here
  00002580           00017 baudrate  equ   .9600           ;define baud rate here
                     00018 ;
  003D0900           00019 fclk      equ   clockrate/4
                     00020 ;*************************************************************
                     00021 ;The value baudconst must be a 8 bit value only
  00000088           00022 baudconst        equ     ((fclk/baudrate)/3 - 2)
                     00023 ;*************************************************************
  00000010           00024 count     equ   0x10          ;used to count tx and rx data bits
  00000011           00025 txreg     equ   0x11          ;used as temp transmit register
  00000011           00026 rcreg     equ   0x11          ;received char is saved here
  00000012           00027 delay     equ   0x12          ;used to time the baud rate.
  00000013           00028 tempa     equ   0x13
  00000010           00029 hi        equ   0x10
  00000011           00030 lo        equ   0x11
  00000015           00031 gpram     equ   0x15
                     00032 ;
                     00033         include "p16c5x.inc"
                     00001         LIST
                     00002 ; P16C5X.INC  Standard Header File, Version 3.30  Microchip Technology, Inc.
                     00224         LIST
                     00034
                     00035 #define _ra1    PORTA,1
                     00036
                     00037 ;
                     00038 #define _tx     PORTA,3
                     00039 #define _rx     PORTA,2
                     00040 ;
0000                 00041         org     0
0000                 00042 start
0000 095A            00043         call    wait
0001 0C0F            00044         movlw   B'00001111'
0002 0025            00045         movwf   PORTA
0003 0C07            00046         movlw   B'00000111'     ;RA3 = output, RA2 = input
0004 0005            00047         tris    PORTA
0005 0066            00048         clrf    PORTB           ;port b all outputs
0006 0040            00049         clrw
0007 0006            00050         tris    PORTB
0008 0CA5            00051         movlw   0xa5            ;test of gpram check if...
```


© 1997 Microchip Technology Inc.

```
0009 0195    00052          xorwf   gpram,W          ;reset was POR or MCLR low
000A 0643    00053          btfsc   STATUS,Z         ;gpram = 0xa5?
000B 0964    00054          call    Mclr             ;yes then do Mclr routine
000C 0CA5    00055          movlw   0xa5             ;else load 0xar in gpram
000D 0035    00056          movwf   gpram
             00057 ;
             00058 ;
000E 0C2F    00059          movlw   0x2f
000F 0033    00060          movwf   tempa
0010 0C38    00061          movlw   B'00111000'
0011 0002    00062          option
0012 0061    00063          clrf    TMR0
0013         00064 S1check
0013 0201    00065          movf    TMR0,W
0014 0643    00066          btfsc   STATUS,Z         ;if S1 pressed then skip
0015 0A13    00067          goto    S1check
             00068 ;
0016         00069 next
0016 02B3    00070          incf    tempa, F
0017 06F3    00071          btfsc   tempa,7
0018 0A61    00072          goto    AllDone
0019 0213    00073          movf    tempa,W
001A 0920    00074          call    transmit
001B 0937    00075          call    receive
001C 0093    00076          subwf   tempa,W
001D 0643    00077          btfsc   STATUS,Z
001E 0A47    00078          goto    fail
001F 0A16    00079          goto    next
             00080 ;
             00081 ;
0020         00082 transmit
0020 0031    00083          movwf   txreg
0021 0465    00084          bcf     _tx              ;send start bit
0022 0C88    00085          movlw   baudconst
0023 0032    00086          movwf   delay
0024 0C09    00087          movlw   .9
0025 0030    00088          movwf   count
0026         00089 txbaudwait
0026 02F2    00090          decfsz  delay, F
0027 0A26    00091          goto    txbaudwait
0028 0C88    00092          movlw   baudconst
0029 0032    00093          movwf   delay
002A 02F0    00094          decfsz  count, F
002B 0A30    00095          goto    SendNextBit
002C 0C09    00096          movlw   .9
002D 0030    00097          movwf   count
002E 0565    00098          bsf     _tx              ;send stop bit
002F 0800    00099          return
0030         00100 SendNextBit
0030 0331    00101          rrf     txreg, F
0031 0703    00102          btfss   STATUS,C
0032 0A35    00103          goto    Setlo
0033 0565    00104          bsf     _tx
0034 0A26    00105          goto    txbaudwait
0035         00106 Setlo
0035 0465    00107          bcf     _tx
0036 0A26    00108          goto    txbaudwait
             00109 ;
             00110 ;
0037         00111 receive
0037 0645    00112        btfsc     _rx
0038 0A37    00113        goto      receive          ;wait for receive
0039         00114 rxbaudwait
0039 02F2    00115          decfsz  delay, F
003A 0A39    00116          goto    rxbaudwait
003B 0C88    00117          movlw   baudconst
```


© 1997 Microchip Technology Inc.

```
003C 0032    00118          movwf   delay
003D 02F0    00119          decfsz  count, F
003E 0A42    00120          goto    RecvNextBit
003F 0C09    00121          movlw   .9
0040 0030    00122          movwf   count
0041 0800    00123          return
0042         00124 RecvNextBit
0042 0403    00125          bcf     STATUS,C
0043 0645    00126          btfsc   _rx
0044 0503    00127          bsf     STATUS,C
0045 0331    00128          rrf     rcreg, F
0046 0A39    00129          goto    rxbaudwait
             00130 ;
0047         00131 fail
0047 0266    00132          comf    PORTB, F
0048 094A    00133          call    halfsec
0049 0A47    00134          goto    fail
004A         00135 halfsec
004A 0070    00136          clrf    hi
004B 0071    00137          clrf    lo
004C         00138 hsloop
004C 0000    00139          nop
004D 0000    00140          nop
004E 0000    00141          nop
004F 0000    00142          nop
0050 0000    00143          nop
0051 0000    00144          nop
0052 0000    00145          nop
0053 0000    00146          nop
0054 0000    00147          nop
0055 02F1    00148          decfsz  lo, F
0056 0A4C    00149          goto    hsloop
0057 02F0    00150          decfsz  hi, F
0058 0A4C    00151          goto    hsloop
0059 0800    00152          return
             00153 ;
005A         00154 wait
005A 0070    00155          clrf    hi
005B 0071    00156          clrf    lo
005C         00157 dly
005C 02F1    00158          decfsz  lo, F
005D 0A5C    00159          goto    dly
005E 02F0    00160          decfsz  hi, F
005F 0A5C    00161          goto    dly
0060 0800    00162          return
             00163
             00164 ;
0061         00165 AllDone
0061 0C55    00166          movlw   0x55
0062 0026    00167          movwf   PORTB
0063 0A63    00168          goto    $
             00169 ;
0064         00170 Mclr
0064 0CAA    00171          movlw   0xaa
0065 0026    00172          movwf   PORTB
0066 0075    00173          clrf    gpram
0067 0000    00174          nop
0068         00175 S3check
0068 0625    00176          btfsc   _ra1
0069 0A68    00177          goto    S3check
006A 0C55    00178          movlw   0x55
006B 0026    00179          movwf   PORTB
006C 0A6C    00180          goto    $
             00181 ;
             00182 ;
01FF         00183          org     0x1ff
```


© 1997 Microchip Technology Inc.

```
01FF 0A00    00184         goto    start
             00185 ;
             00186         end

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   110
Program Memory Words Free:   402


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Modifying PIC16C54A Code for the PIC16C58A

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

Code written for a PIC16C54A device can be easily modified and used in a PIC16C58A. Therefore, a PIC16C58A may be used in place of a PIC16C54A. The PIC16C58A and the PIC16C54A are pin-compatible devices. Also, the PIC16C58A has more than enough memory (program and data) to support PIC16C54A code.

Device	Pins	EPROM/ROM	RAM
PIC16C54A	18	512	25
PIC16C58A	18	2048	73

PROGRAM MEMORY

The PIC16C54A has 512 words of EPROM program memory, while the PIC16C58A has 2048 words. Figure 1 and Figure 2 depict the memory organization for each of these devices. For this application, Page 0 of the PIC16C58A On-chip Program Memory will be used in place of the PIC16C54A On-chip Program Memory. However, the Reset vector location will change from 1FFh for the PIC16C54A to 7FFh for the PIC16C58A. (See the discussion on Reset in the next section.)

The PC, or program counter, is used on both devices to access program memory locations. Eight of the nine bits required to access the 512 words of program memory in the PIC16C54A are provided by the PCL file register (Figure 3). The most significant 9th bit is provided from the instruction word during a GOTO instruction. A CALL instruction, on the other hand, forces a '0' into the most significant 9th bit of the PC. Hence, all subroutines must reside in the top half (00h to FFh) of the program memory.

On the PIC16C58A, bits 0 through 8 of the PC operate the same as bits 0 through 8 of the PIC16C54A PC for GOTO and CALL instructions (Figure 4). The PIC16C58A PC differs from the PIC16C54A PC in the addition of two most significant bits. These 10th and 11th bits are provided by the PA0 and PA1 values of the STATUS register. The bits signify the program memory page for the PIC16C58A (i.e., 00 = Page 0, 01 = Page 1, etc.).

After a power-up reset, the PA1:PA0 bits are reset to '0'. So, if these two bits are not modified by the PIC16C54A program, the same program will correctly access the Page 0 program memory space on a PIC16C58A.

FIGURE 1: PIC16C54A PROGRAM MEMORY MAP AND STACK

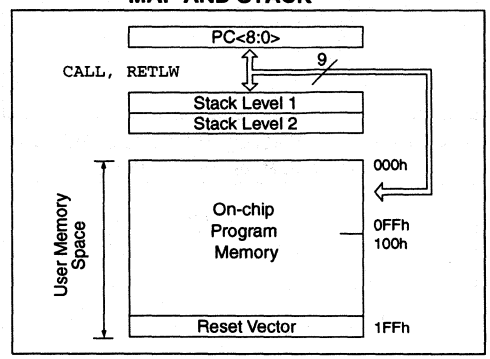
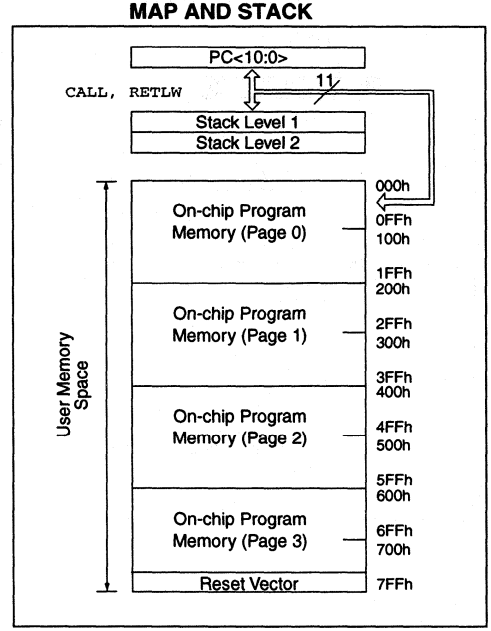


FIGURE 2: PIC16C58A PROGRAM MEMORY MAP AND STACK



© 1997 Microchip Technology Inc.

FIGURE 3: LOADING OF PIC16C54A PC FOR BRANCH INSTRUCTIONS

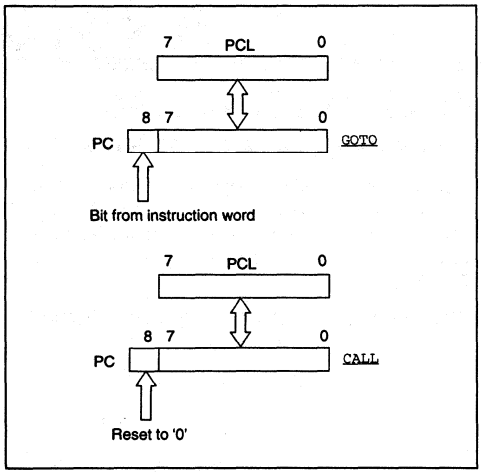
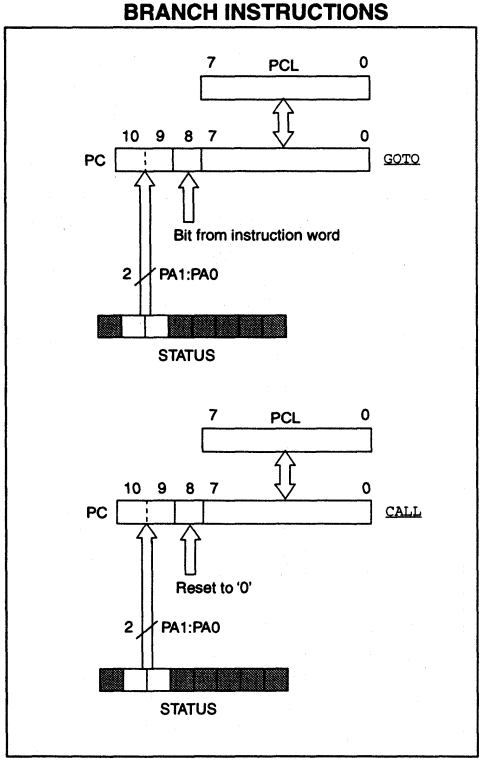


FIGURE 4: LOADING OF PIC16C58A PC FOR BRANCH INSTRUCTIONS



Reset Address

The reset address on the PIC16C54A is at location 1FFh. Normally, a "GOTO Start" instruction is placed at this location, where Start is the beginning of the program. If the user has used such an instruction, then in order for the program to operate correctly on a PIC16C58A the "GOTO Start" instruction should be moved to location 7FFh, the reset address for the PIC16C58A.

Operation in a Noisy Environment

What happens if the PA1 or PA0 bit value gets corrupted due to noise or some other external event? The PA1:PA0 bits are not directly mapped into PC<11:9>, but the PA1:PA0 bit values are loaded into PC<11:9> during a CALL or GOTO instruction. So, if the PA1:PA0 bit values get corrupted (ex: PA1:PA0 = 11 instead of 00), the program will go to a program memory page other than 0 when a CALL or GOTO instruction occurs. The program will then start executing "unimplemented" code space.

Normally the user would program "unimplemented" code space with the default FFFh value (XORLW FFh instruction). The PC would increment sequentially until it hit the last location (7FFh) or the Reset address, which is normally recommended to be a "GOTO Start" instruction. However, since the value in PA1:PA0 is still corrupted (PA1:PA0 = 11 in our example), the program execution will still be in Page 3. In other words, the program would get locked in an endless loop.

In order to avoid the above situation, each "unimplemented" page should be filled with the instruction "GOTO Recover". The Recover subroutines (Three will be required for the three (3) unimplemented pages of the PIC16C58A) should clear the PA1:PA0 bits in the STATUS register and execute a "GOTO Start" instruction.

EXAMPLE 1: RECOVER CODE

```
RECOVER  BCF STATUS, PA1
         BCF STATUS, PA0
         GOTO START
```

Please note that operation in a noisy environment is very application-dependent and may not affect the majority of the users. However, due to the presence of extra code space on the PIC16C58A, additional steps may have to be taken as mentioned above. Please check Appendix A for an example of the code implementation.

© 1997 Microchip Technology Inc.

DATA MEMORY

Figure 5 and Figure 6 depict the data memory registers for the PIC16C54A and the PIC16C58A. Banks 1, 2 and 3, which are not present in the PIC16C54A, are R/W memory RAM locations on the PIC16C58A. These locations are accessed using the FSR<6:5> bits. On the PIC16C54A, these bit values are ignored. However, they are not cleared during a power-up reset on any PIC16C5XA device. Therefore, the FSR<6:5> bits on the PIC16C58A should be cleared at the very start of the program and always maintained in that manner during the course of the program.

Operating in a Noisy Environment

The FSR<6:5> bits can get corrupted due to noise or some external conditions just like the PA1:PA0 bits of program memory. As part of that recovery, the user should also clear the FSR<6:5> bits. Please see Appendix A for a code example.

SUMMARY

Code transfer from a PIC16C54A to a PIC16C58A may be accomplished easily by remembering the following points:

- The reset address has to be re-located from 1FFh to 7FFh.
- The PA1:PA0 bits of the STATUS register must be kept cleared.
- The FSR<6:5> bits must be kept cleared.

FIGURE 5: PIC16C54A REGISTER FILE MAP

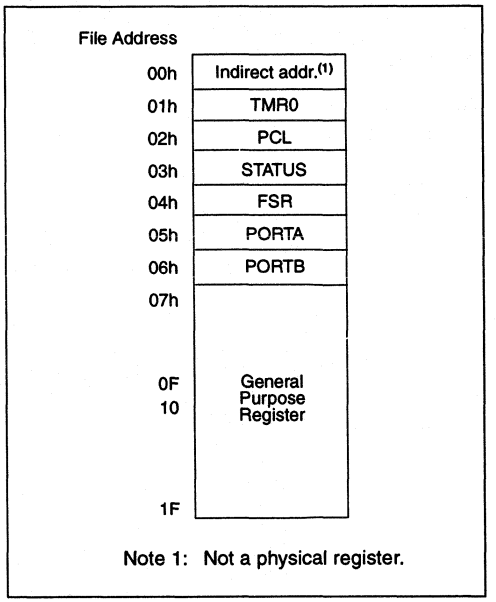
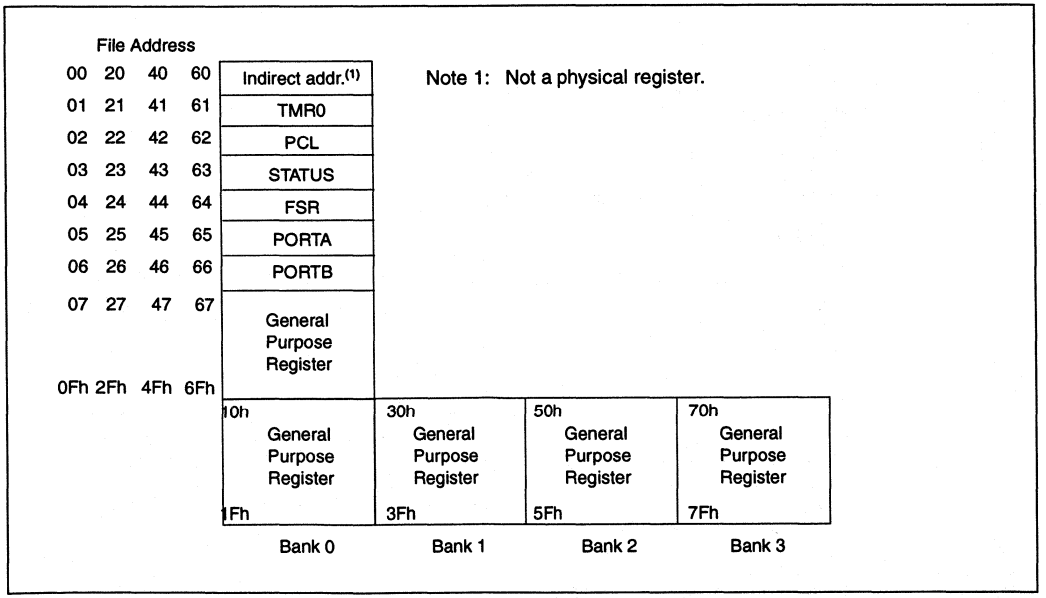


FIGURE 6: PIC16C58A REGISTER FILE MAP



Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.21 Released          P54TO58.ASM   8-25-1995  12:33:54         PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;This program is a code example of a program written for a PIC16C54A
                     00002 ;and modified for use in a PIC16C58A.
                     00003 ;
                     00004 ;In this program the following modifications have been done:
                     00005 ;1. The reset address is relocated from 0x1ff to 0x7ff
                     00006 ;2. The FSR<6,5> bits are initialized to 0 at the start of the program
                     00007 ;3. The Recover subroutine implemented in each "un-implemented"
                     00008 ;   page of the PIC16C58A.
                     00009 ;
                     00010 ;                                              by Stan D'Souza 8/29/95.
                     00011 ;
                     00012 ;
                     00013         list p=16C58A, f=inhx8m
                     00014 ;
                     00015 ;This program written for the PIC16C54A and operates on the PICDEM1
                     00016 ;demo board. Program has been modified to work on a PIC16C58A.
                     00017 ;The program blinks each of the 8 LEDs on PortB twice, then moves to the
                     00018 ;next LED and so on. If anytime the RA1 key is pressed, the direction
                     00019 ;of rotation is changed.
                     00020 ;
                     00021 ;
  0006               00022 PORTB   equ             6
  0005               00023 PORTA   equ             5
  0003               00024 STATUS  equ             3
  0004               00025 FSR     equ             4
  0005               00026 PA0     equ             5
  0006               00027 PA1     equ             6
  0000               00028 W       equ             0
  0001               00029 F       equ             1
  0000               00030 C       equ             0
  001B               00031 pb_buf  equ             0x1b
  001C               00032 count   equ             0x1c
  001D               00033 templ   equ             0x1d
  001E               00034 temph   equ             0x1e
  001F               00035 flag    equ             0x1f
                     00036 #define KeyPressed      flag,7
                     00037 #define KeyReleased     flag,6
                     00038 #define LtoR            flag,5
                     00039 ;
                     00040 ;*****************************************************************
                     00041 ;       org     0x1ff           ;reset vector for PIC16C54A, comment out
                     00042                                 ;when moving to a PIC16C58A
                     00043 ;*****************************************************************
07FF                 00044         org     0x7ff           ;added as PIC16C58A reset vector
07FF 0A10            00045         goto    Start
                     00046 ;*****************************************************************
                     00047 ;The fill command is used to load the unused 1.5K EPROM space on the
                     00048 ;PIC16C58A with a "goto Recover" subroutine. If the program ever
                     00049 ;enters the unused 1.5K space, this command will immediately and
                     00050 ;automatically cause the program execution to go back to the first page
                     00051 ;at address 0x00 to 0x1ff.
0200                 00052         org     0x200
0200                 00053 Recover1
0200 04C3            00054         bcf     STATUS,PA1
0201 04A3            00055         bcf     STATUS,PA0
```


© 1997 Microchip Technology Inc.

```
0202 0064          00056          clrf    FSR
0203 0A10          00057          goto    Start
0204 0A00          00058          FILL    (goto Recover1), (0x3ff - $ + 1)
                   00059          ;Fill all unused locations in page 1 with goto Recover1
                   00060 ;
                   00061 ;
0400               00062          org     0x400
0400               00063 Recover2
0400 04C3          00064          bcf     STATUS,PA1
0401 04A3          00065          bcf     STATUS,PA0
0402 0064          00066          clrf    FSR
0403 0A10          00067          goto    Start
0404 0A00          00068          FILL    (goto Recover2), (0x5ff - $ + 1)
                   00069          ;Fill all unused locations in page 2 with goto Recover2
                   00070 ;
                   00071 ;
                   00072 ;
0600               00073          org     0x600
0600               00074 Recover3
0600 04C3          00075          bcf     STATUS,PA1
0601 04A3          00076          bcf     STATUS,PA0
0602 0064          00077          clrf    FSR
0603 0A10          00078          goto    Start
0604 0A00          00079          FILL    (goto Recover3), (0x7fe - $ + 1)
                   00080          ;Fill all unused locations in page 3 with goto Recover3
                   00081 ;
                   00082 ;
0010               00083          org     0x10
0010               00084 Start
                   00085 ;*****************************************************************
0010 0064          00086          clrf    FSR             ;initialize the FSR register
                   00087                                  ;added for the PIC16C58A compatibility
                   00088 ;*****************************************************************
0011 0040          00089          clrw
0012 0026          00090          movwf   PORTB           ;set Port B as output and low
0013 0006          00091          tris    PORTB           ;       /
0014 007B          00092          clrf    pb_buf          ;clear buffer
0015 051B          00093          bsf     pb_buf,0        ;set up the first blink
0016 007F          00094          clrf    flag            ;clr flags
0017 05DF          00095          bsf     KeyReleased     ;       /
0018               00096 Repeat
0018 021B          00097          movf    pb_buf,W
0019 0026          00098          movwf   PORTB
001A 092F          00099          call    delay500        ;delay for 500mS
001B 0066          00100          clrf    PORTB
001C 092F          00101          call    delay500        ;
001D 06FF          00102          btfsc   KeyPressed      ;if no key then skip
001E 0A46          00103          goto    ChangeDirection ;else change direction
001F 021B          00104          movf    pb_buf,W
0020 0026          00105          movwf   PORTB
0021 092F          00106          call    delay500        ;delay for 500mS
0022 0066          00107          clrf    PORTB
0023 092F          00108          call    delay500        ;
0024 06FF          00109          btfsc   KeyPressed      ;if no key then skip
0025 0A46          00110          goto    ChangeDirection
0026               00111 Rotate
0026 0403          00112          bcf     STATUS,C        ;clr carry
0027               00113 RotAgain
0027 07BF          00114          btfss   LtoR            ;left to right?
0028 0A2D          00115          goto    RotateLeft      ;rotate Port B
0029 033B          00116          rrf     pb_buf,F        ;rotate Port B
002A               00117 Done
002A 0703          00118          btfss   STATUS,C        ;carry over?
002B 0A18          00119          goto    Repeat          ;no then do again
002C 0A27          00120          goto    RotAgain        ;rotate again
002D               00121 RotateLeft
```

```
002D 037B        00122         rlf     pb_buf,F        ;rotate left
002E 0A2A        00123         goto    Done            ;check
                 00124 ;
002F             00125 delay500
002F 0C19        00126         movlw   .25             ;reload count
0030 003C        00127         movwf   count           ;       /
0031             00128 dly5001p
0031 06DF        00129         btfsc   KeyReleased     ;key released?
0032 0938        00130         call    CheckReleased   ;no then check?
0033 093E        00131         call    CheckPressed    ;else check if pressed?
0034 094E        00132         call    delay20         ;delay for 20mS
0035 02FC        00133         decfsz  count           ;see if delay over
0036 0A31        00134         goto    dly5001p        ;no then loop
0037 0800        00135         retlw   0               ;return
                 00136 ;
0038             00137 CheckReleased
0038 0625        00138         btfsc   PORTA,1         ;RA1 low?
0039 0A3B        00139         goto    RelAgain        ;no then check again
003A 0800        00140         retlw   0               ;go back
003B             00141 RelAgain
003B 0625        00142         btfsc   PORTA,1         ;RA1 low?
003C 05DF        00143         bsf     KeyReleased     ;no then key released
003D 0800        00144         retlw   0
                 00145 ;
003E             00146 CheckPressed
003E 06FF        00147         btfsc   KeyPressed      ;flag already set?
003F 0800        00148         retlw   0               ;yes then return
0040 0725        00149         btfss   PORTA,1         ;see if key low?
0041 0A43        00150         goto    PressAgain      ;check again
0042 0800        00151         retlw   0               ;else go back
0043             00152 PressAgain
0043 0725        00153         btfss   PORTA,1         ;see if low
0044 05FF        00154         bsf     KeyPressed      ;yes then set flag
0045 0800        00155         retlw   0
                 00156 ;
0046             00157 ChangeDirection
0046 04FF        00158         bcf     KeyPressed      ;key serviced
0047 04DF        00159         bcf     KeyReleased     ;see if key released
0048 07BF        00160         btfss   LtoR            ;check if Left to Rt.
0049 0A4C        00161         goto    TurnLeft        ;make it go left
004A 04BF        00162         bcf     LtoR
004B 0A26        00163         goto    Rotate
004C             00164 TurnLeft
004C 05BF        00165         bsf     LtoR
004D 0A26        00166         goto    Rotate
                 00167
                 00168 ;
                 00169 ;Delay loop for 20 mS
004E             00170 delay20
004E 0C14        00171         movlw   .20             ;load temph
004F 003E        00172         movwf   temph
0050             00173 dly4
0050 0CC8        00174         movlw   .200            ;load templ
0051 003D        00175         movwf   templ
0052             00176 dly
0052 0000        00177         nop
0053 0000        00178         nop
0054 02FD        00179         decfsz  templ
0055 0A52        00180         goto    dly
0056 02FE        00181         decfsz  temph
0057 0A50        00182         goto    dly4
0058 0800        00183         retlw   0
                 00184 ;
                 00185 ;
                 00186 ;
                 00187         end
```


© 1997 Microchip Technology Inc.

```
All other memory blocks unused.


Errors    :     0
Warnings  :     0
Messages  :     0
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Code Development for the PIC16C52

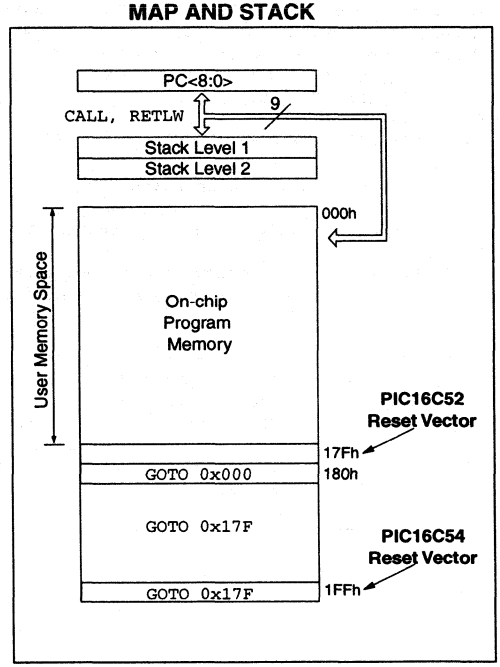
Author: *Stan D'Souza*
Microchip Technolgoy Inc.

CODE DEVELOPMENT PROCEDURE

To develop code on a PIC16C52 device, a PIC16C54/JW part can be used. In order to use a PIC16C54/JW device, the following conditions should be followed:

1. A PIC16C54/JW has 512 words (0x1FF) of program memory, whereas a PIC16C52 has 384 bytes (0x17F) of program memory. The "extra" locations should be programmed as follows:
   a) Location 0x180 should be programmed with a "`GOTO 0x00`" instruction
   b) Locations 0x181 through 0x1FF should be programmed with a "`GOTO 0x17F`" instruction.
   See Figure 1 for details, and for a standard "skeleton" program, use the code in Example 1.
2. The PIC16C54/JW has a Watchdog Timer (WDT), while the PIC16C52 does not. Therefore, the WDT on the PIC16C54/JW must be disabled. The "__CONFIG" command used in Example 1 disables the WDT.
3. The PIC16C54/JW has four oscillator modes, while the PIC16C52 has two. Therefore, only RC and XT oscillator modes should be used. The "__CONFIG" command used in Example 1 takes care of programming the XT Oscillator mode, if required.

FIGURE 1: PIC16C54 PROGRAM MEMORY MAP AND STACK



© 1997 Microchip Technology Inc.

EXAMPLE 1: CONVERSION CODE

```
;This test file "skeleton" depicts a standard format, which
;a user can use when programming a PIC16C54-JW device to
;"emulate" a PIC16C52, during code development
;
        list p=16c54, f=inhx8m
;
;include the header file from the correct directory, please
;verify the correct dos path for the header file "P16C5X.INC"
        include "c:\pictools\mpasm\P16C5X.INC"
;Define the configuration bits. Please note that the PIC16C52
;does not support the WDT, so _WDT_OFF should be selected.
;Also only RC and XT Oscillator modes are supported on the PIC16C52.
;Code protect is supported on the PIC16C52.
        __CONFIG  _WDT_OFF & _XT_OSC      ;for XT mode only
;       __CONFIG  _WDT_OFF                ;for RC mode
;
        org     0x00
Start
        ;User's code should reside from 0x00 to 0x17F
        ;
        org     0x17f   ;reset vector for PIC16C52
;
;A "goto Start" instruction is recommended to be programmed at the reset
;vector, however the user may elect to program any code at this
;location.
;
;*************************************************************************
;The code below MUST be programmed as indicated, in order for a
;PIC16C54-JW to work as a PIC16C52.
        org     0x180   ;program location not present in PIC16C52
        goto    0x00    ;roll over to location 0
;fill remaining locations with goto reset vector for PIC16C52
        FILL    (GOTO 0x17F), (0x1FF - $ + 1)
;
;*************************************************************************
        end
```


© 1997 Microchip Technology Inc.

D/A Conversion Using PWM and R-2R Ladders to Generate Sine and DTMF Waveforms

Authors: Rob Stein and John Day
Microchip Technology Inc.

INTRODUCTION

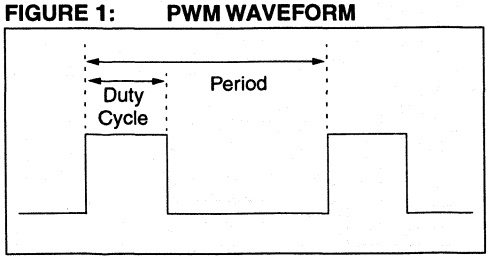
Many embedded applications require the generation of analog signals. Although separate D/A converter IC's exist on the market today, their extra price makes them prohibitive in cost sensitive embedded control designs. The following application note describes two DAC designs for generation of complex analog waveforms.: (PWM and R-2R Ladder). Both the Pulse Width Modulation (PWM) and resistor ladders require only a few external components to create a D/A converter with the PICmicro™ microcontroller series.

These techniques can be used to generate dual-tone multiplexed frequencies (DTMF) for telephone dialing, controlling the speed of a motor, generating sound and complex waveforms, and generating variable voltages in a power supply.

PULSE WIDTH MODULATION

Pulse Width Modulation (PWM) involves the generation of a series of pulses at a fixed period and frequency. The duty cycle defines the width of each pulse which is varied to generate waveforms. A simple low pass filter is then used to generate an output voltage directly proportional to the average time spent in the HIGH state (i.e., 50% duty cycle is equal to 2.5 volts when V_{DD} = 5.0V).

FIGURE 1: PWM WAVEFORM

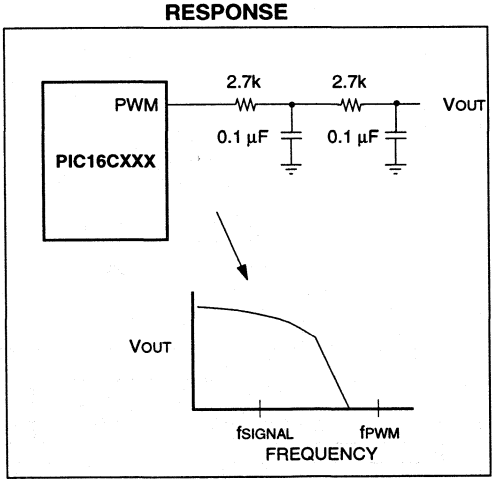


This D/A technique is used to produce slow moving analog outputs (from 0 to 100's of Hz). The software `SINE.ASM`, written for the PIC16C620, produces a 60 Hz sine function at a 5-bit resolution (32 steps). For each step that is generated the software allows ~ 10 PWM cycles for the low pass to settle. The number of PWM cycles for the low pass to settle varies from one application to another. The designer should experiment with this value to find the most optimal number for the application. These two criteria (sine frequency and # of steps) lead to the required PWM frequency:

$$\begin{aligned} PWM_Freq &= (Sine_freq) \bullet (\#\ of\ steps) \bullet 10 \\ &= 60 \bullet 32 \bullet 10 \\ &= 19.2\ kHz \end{aligned}$$

A simple RC low pass filter is the only external component needed to complete the circuit. A two pole filter is used in the example to pass the 60 Hz signal while eliminating the step (1920 Hz) and the PWM (19 khz) frequencies.

FIGURE 2: RC FILTER FREQUENCY RESPONSE



For this application note a software implementation was used to generate the PWM due to it's inherent lower cost. If greater resolution or frequency is required, a PICmicro family microcontroller with a hardware PWM should be used.

The TMR0 interrupt is used to generate precise PWM transitions. The time to service this interrupt will limit the minimum and maximum duty cycle for the PWM and hence the min/max voltage that can be produced. Therefore the interrupt service routine has been written to use as few cycles as possible.

A table lookup technique is used to generate one period of a sine function. Other waveforms, such as DC voltage ramps, triangles, sounds etc., can be produced

© 1997 Microchip Technology Inc.

from this technique as well. The number of table entries is equal to the resolution required. The designer should carefully investigate the tradeoffs between PWM frequency, RC settling time, waveform resolution, and waveform frequency.

SINE.ASM SOURCE CODE

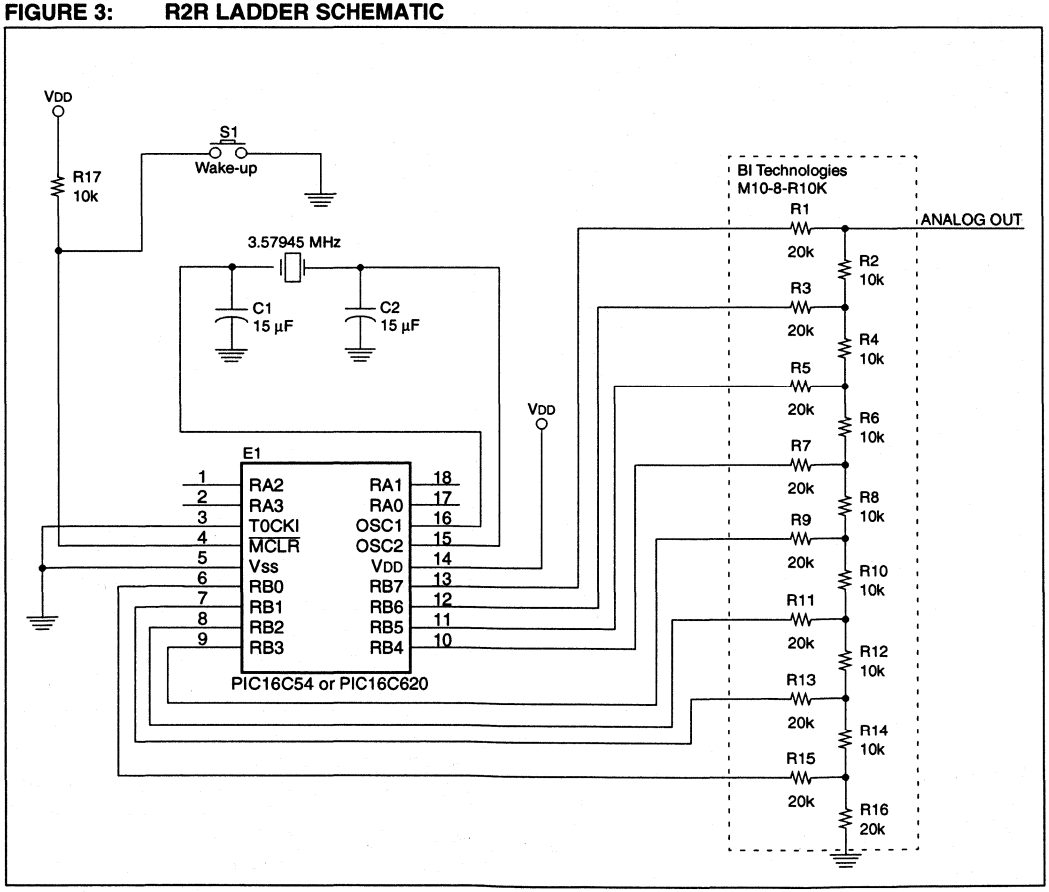
The `SINE.ASM` code example consists of four major code segments:

1. A main routine for counting and timing the 32 steps of the sine wave.
2. The `SetPWM` subroutine to pass a new PWM high and low counter value to the interrupt service routine at each step transition.
3. A table lookup to hold the 32 separate steps for the sine wave.
4. An interrupt service routine which creates the precise timing for the high and low PWM cycles.

R-2R LADDER

Many D/A converters are constructed with an internal resistor network driven by digital output drivers. The resistors are wired in a ladder fashion, with resistors of the same value being used for the rails, and resistors with 2x that value being used for the digital port outputs. Figure 3 shows the connection of a R2R ladder.

FIGURE 3: R2R LADDER SCHEMATIC



© 1997 Microchip Technology Inc.

Implementation of a R2R ladder can be very inexpensive. BI Technologies sells a single SIP package for 8-bit and 10-bit R2R ladder networks. 8-bit R2R networks can also be manually constructed using 16 discrete resistors. Either way, R2R ladder networks have low EMI emissions, and reduced high frequency harmonics, eliminating the need for a low pass filter in most designs.

Microchip's microcontrollers have I/O pins with very low drive impedance (<75 Ohms at 5V) for both driving logic '1' and '0'. This allows direct connection to the R2R ladder without any external transistor drivers. To achieve 1 LSb accuracy using an 8-bit D/A, it is important to select resistors whose source impedance (in this case, 20k) is ≥ 256 times the driver source impedance.

To test the linearity of this design a +5V power supply was used. Each bit pattern was selected, then the output voltage was measured. The results show the transfer function in Figure 4.

SINE WAVE FUNCTIONS

The simplest way to generate sine waves is through a lookup table. One simply needs to determine the maximum allowable distortion along with the maximum operating frequency. Using these two criteria, the number of sample points needed can be determined:

Sample points = 100 / (% allowable voltage step inaccuracy)

Sample time = (Sample Points) • (Frequency)

Example:

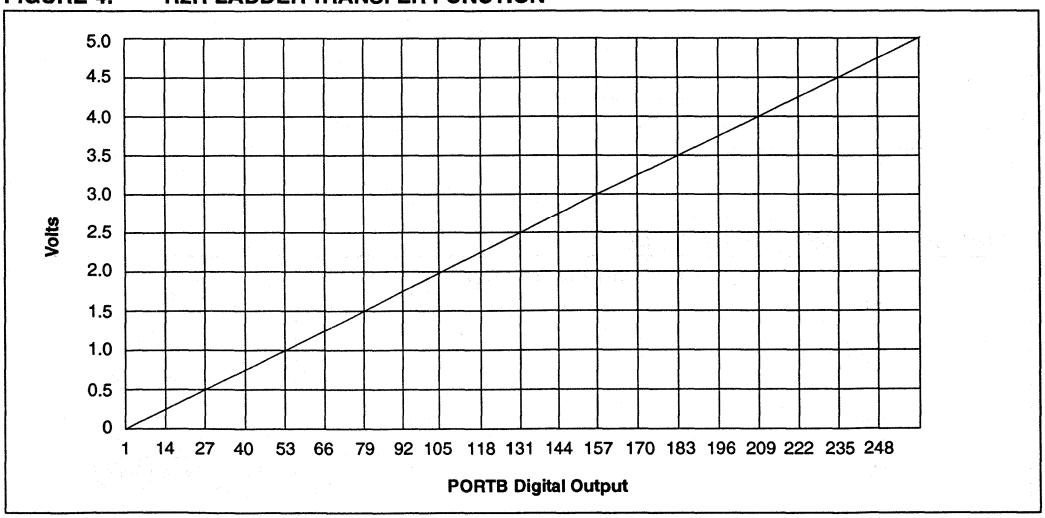
2 kHz Sine wave, <8% voltage step inaccuracy

Sample points = 100/5 = 20 points

Sample time = 20 • 2,000 = 40 kHz = 50 µs/Sample

To generate a single sine wave, the user simply needs to create a lookup table with the correct number of sample points. Next the software steps through each sample point and moves this value to the PORT output register, sending the correct voltage out of the R2R ladder. It is critical to ensure that the every sample point value be moved to the port output register using the same timing interval.

FIGURE 4: R2R LADDER TRANSFER FUNCTION



DTMF GENERATION

Many telecom applications, such as auto dialers, telephone keypads and security systems, require DTMF transmission for dialing and data transmission. Using the R2R ladder and seven sine lookup tables, an entire 12-key keypad of DTMF patterns can be generated with a 3.579545 MHz color burst crystal.

The DTMF standard was originally developed by Bell Labs for use by AT&T in telephone systems. There are several specifications that have fallen out of the original standard which come from AT&T, CEPT, CCITT, etc,. The variations from one standard to another are typically deviations in frequency tolerance, power, power difference between two tones, and speech immunity. The CCITT standard is located in Recommendations Q.23 and Q.24 in Section 4.3 of the CCITT Red Book, Volume VI, Fascicle VI.1. The other standards are listed in the References section at the end of this chapter.

A telephone keypad is broken up into 4 rows and 3 columns for a total of 12 keys. Each row and column is represented by a frequency; therefore, each key is uniquely represented by the sum of a row and column as follows:

TABLE 1: 12-KEY KEYPAD FREQUENCY

	1209	1336	1477
697	1	2	3
770	4	5	6
852	7	8	9
941	*	0	#

As an example, the '1' key is represented by a tone of 697 and 1209 Hz simultaneously.

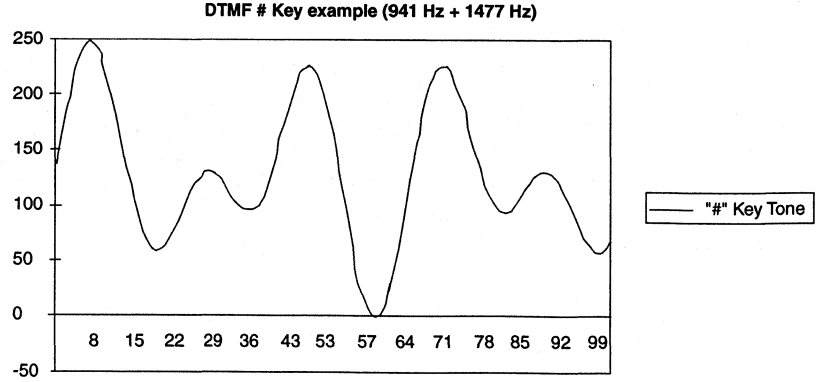
DTMF.ASM SOURCE CODE EXAMPLE

The source code `DTMF.ASM` shows how to generate the complete DTMF table for all keys. The complete DTMF generation algorithm requires only 285 words of code space, including all 7 sine lookup tables. The first 220 words of code space is used for the 7 sine lookup tables. Each sine wave lookup table ends with a value of 127 decimal to show the end of the sine wave pattern. The next section contains a look-up table for each key number which is used to look-up the sine wave pattern frequency number.

The `senddtmf` routine is used to beep the DTMF for 320 ms. Prior to calling the `senddtmf` routine, the key's number is loaded into the W register. The sine wave addresses for that key are first calculated and the `WAVEABASE` and `WAVEBBASE` registers are loaded with the sine addresses. Next, each sine wave sample point is looked up and the sum of the two sine waves is moved to PORTB. Lastly, the pointer is reset to the sine base address if the value is equal to 127 decimal.

The test program loads each key into W and then calls `senddtmf`, showing all possible key patterns. The following figure shows a trace of the # key tones:

FIGURE 5: DTMF # KEY EXAMPLE



Note: This is a basic circuit. It is left to the designer to ensure that the final application conforms to the DTMF standard.

© 1997 Microchip Technology Inc.

CONCLUSION

Using either PWM or R-2R ladder design techniques, many PICmicro microcontroller embedded designs will benefit from their low cost and component count. The following table compares the characteristics of the software and hardware PWM, R-2R ladder and external DAC design alternatives and their respective advantages and disadvantages.

TABLE 2: COST / PERFORMANCE TRADEOFF CHART

Cost / Performance Tradeoffs				
	S/W PWM	**H/W PWM**	**R2R Ladder**	**External DAC**
Cost	Low	Medium	Low	High
# of External Components	Low (1)	Low (1)	Low/Medium (2)	Low
I/Os Required	One	One	Eight (3)	Two/Nine (4)
Accuracy	Good (5)	Good (5)	Good (6)	Good/Excellent (7)
Resolution	Excellent (8)	Excellent (8)	Excellent (9)	Excellent (9)
Bandwidth	Low (10)	Low/Medium(10)	High	High
Harmonic Distortion	High	Medium	Low	Low

Note 1: One capacitor and one resistor for the low pass filter.
2: Multiply the required resolution times two for the number of external resistors or purchase one resistor SIP (16 resistors are required for 8-bit resolution).
3: The number of I/O's are directly proportional to resolution (8 I/O's for 8-bit resolution).
4: Separate DAC IC's are available with serial or parallel communication.
5: Absolute accuracy is dependent upon supply voltage.
6: Absolute accuracy is dependent upon supply voltage and resistor ladder tolerance.
7: Depends upon D/A reference source (supply voltage or a separate voltage reference).
8: Resolution is a dependent upon the PWM resolution.
9: Resolution is dependent upon the number of resistors in the ladder.
10: The waveform bandwidth is limited by the maximum PWM frequency and the settling time for the low pass filter. For an 8-bit resolution PWM the maximum frequency of operation is 19.5 kHz for a software PWM, and 80 kHz for a hardware PWM.

REFERENCE

Paul Horowitz and Winfield Hill:
"The Art of Electronics",
Cambridge University Press, New York 1989.

Contains a excellent practical description of analog filter design (as well as numerous other useful electronic subjects).

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40.01 Intermediate      DTMF.ASM   3-31-1997  11:00:08         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ; Filename: DTMF.ASM
                     00002 ; ********************************************
                     00003 ; * Author:   John Day                       *
                     00004 ; *           Sr. Field Applications Engineer  *
                     00005 ; *           Microchip Technology             *
                     00006 ; * Revision: 1.1                            *
                     00007 ; * Date      December 20, 1995              *
                     00008 ; * Part:     PIC16C54                       *
                     00009 ; * Compiled using MPASM V1.40               *
                     00010 ; ********************************************
                     00011 ; * Include files:                           *
                     00012 ; *           NONE (used by DTMF.ASM)        *
                     00013 ; ********************************************
                     00014 ; * Fuses:    OSC:  XT (3.579545 Mhz xtal)   *
                     00015 ; *           WDT:  OFF                      *
                     00016 ; *            CP:  OFF                      *
                     00017 ; ************************************************************************
                     00018 ; * This program uses and external R2R ladder network to generate complete *
                     00019 ; * DTMF dial tones used for telphone dialing.                             *
                     00020 ; ************************************************************************
                     00021 ; * Program Memory:                                                       *
                     00022 ; *            220 Words - sine wave look-up table (7 sine waves total)   *
                     00023 ; *             25 Words - keypad sine address matrix look-up            *
                     00024 ; *             37 Words - DTMF sine wave base initialization/generation *
                     00025 ; *              3 Words - Initialization                                *
                     00026 ; *             25 Words - Test sample code                              *
                     00027 ; * RAM Memory:                                                          *
                     00028 ; *              8 Bytes                                                 *
                     00029 ; ************************************************************************
                     00030         list     p=16C54, r=dec
                     00031         #include <p16c5x.inc>
                     00001         LIST
                     00002 ; P16C5X.INC  Standard Header File, Version 3.30 Microchip Technology, Inc.
                     00224         LIST
0FFF 0FF9            00032         __CONFIG _XT_OSC&_WDT_OFF&_CP_OFF
                     00033
  00000010           00034 WAVEABASE       EQU     10h     ; Base address of sine A waveform
  00000011           00035 POINTERA        EQU     11h     ; Pointer to current position in sine A
  00000012           00036 WAVEBBASE       EQU     12h     ; Base address of sine B waveform
  00000013           00037 POINTERB        EQU     13h     ; Pointer to current position in sine B
  00000014           00038 NEXTVALUE       EQU     14h     ; Sum register to store Sine A + Sine B
  00000015           00039 SINECOUNT       EQU     15h     ; LSB counter for time to output DTMF
  00000016           00040 SINECOUNTH      EQU     16h     ; MSB counter for time to output DTMF
  00000017           00041 TEMP            EQU     17h     ; Temporary storage
  0000007F           00042 ENDSINE         EQU     .127    ; Value to show the end of a sine table
                     00043 ; ************************************************************************
                     00044 ; * sinelookup                                                           *
                     00045 ; * This is the look-up table for the (4x3) keypad matrix sine wave table  *
                     00046 ; * There are (7) sine waves stored here and adding any two from the       *
                     00047 ; * matrix will product a DTMF signal for the appropriate key             *
                     00048 ; * Crystal Frequency:  3.579545 Mhz                                      *
                     00049 ; * Instructions/Loop:  35                                               *
                     00050 ; * Base Frequency:     1209 1336 1477  697  770  852  941 Hz           *
                     00051 ; * Actual Frequency:   1217 1345 1475  691  774  852  946 Hz           *
```


© 1997 Microchip Technology Inc.

```
                00052 ; * Error                0.7  0.7 -0.1 -0.9  0.5    0  0.5 %              *
                00053 ; * Num Table Entries:    21   19   52   37   33   30   27                *
                00054 ; * Total Table Entries: 219                                              *
                00055 ; * Program Memory:                                                       *
                00056 ; *         220 Words - Used for (7) sine look-up entries               *
                00057 ; * RAM Memory:                                                          *
                00058 ; *     NONE - Look-up table only                                        *
                00059 ; ***************************************************************************
0000            00060 sinelookup                ; Used as address table to call look-up table
0000 01E2       00061         addwf     PCL,F   ; Add sine offset to PC to jump into table
0001            00062 sineoffset                ; Used to calculate offset value address
0001            00063 sinerow1                  ; Address for sine wave in row 1
0001 0895       00064         retlw     149
0002 08AA       00065         retlw     170
0003 08BE       00066         retlw     190
0004 08D0       00067         retlw     208
0005 08E0       00068         retlw     224
0006 08EC       00069         retlw     236
0007 08F6       00070         retlw     246
0008 08FD       00071         retlw     253
0009 08FF       00072         retlw     255
000A 08FE       00073         retlw     254
000B 08FA       00074         retlw     250
000C 08F2       00075         retlw     242
000D 08E6       00076         retlw     230
000E 08D8       00077         retlw     216
000F 08C7       00078         retlw     199
0010 08B4       00079         retlw     180
0011 08A0       00080         retlw     160
0012 088A       00081         retlw     138
0013 0875       00082         retlw     117
0014 085F       00083         retlw      95
0015 084B       00084         retlw      75
0016 0838       00085         retlw      56
0017 0827       00086         retlw      39
0018 0819       00087         retlw      25
0019 080D       00088         retlw      13
001A 0805       00089         retlw       5
001B 0801       00090         retlw       1
001C 0800       00091         retlw       0
001D 0802       00092         retlw       2
001E 0809       00093         retlw       9
001F 0813       00094         retlw      19
0020 081F       00095         retlw      31
0021 082F       00096         retlw      47
0022 0841       00097         retlw      65
0023 0855       00098         retlw      85
0024 086A       00099         retlw     106
0025 087F       00100         retlw     127     ; End of this sine wave
0026            00101 sinerow2                  ; Address for sine wave in row 2
0026 0898       00102         retlw     152
0027 08AF       00103         retlw     175
0028 08C5       00104         retlw     197
0029 08D8       00105         retlw     216
002A 08E8       00106         retlw     232
002B 08F4       00107         retlw     244
002C 08FC       00108         retlw     252
002D 08FF       00109         retlw     255
002E 08FE       00110         retlw     254
002F 08F8       00111         retlw     248
0030 08EE       00112         retlw     238
0031 08E0       00113         retlw     224
0032 08CF       00114         retlw     207
0033 08BA       00115         retlw     186
0034 08A4       00116         retlw     164
0035 088C       00117         retlw     140
```


© 1997 Microchip Technology Inc.

```
0036 0873     00118          retlw      115
0037 085B     00119          retlw       91
0038 0845     00120          retlw       69
0039 0830     00121          retlw       48
003A 081F     00122          retlw       31
003B 0811     00123          retlw       17
003C 0807     00124          retlw        7
003D 0801     00125          retlw        1
003E 0800     00126          retlw        0
003F 0803     00127          retlw        3
0040 080B     00128          retlw       11
0041 0817     00129          retlw       23
0042 0827     00130          retlw       39
0043 083A     00131          retlw       58
0044 0850     00132          retlw       80
0045 0867     00133          retlw      103
0046 087F     00134          retlw      127      ; End of this sine wave
0047          00135 sinerow3                     ; Address for sine wave in row 3
0047 089A     00136          retlw      154
0048 08B4     00137          retlw      180
0049 08CB     00138          retlw      203
004A 08DF     00139          retlw      223
004B 08EE     00140          retlw      238
004C 08F9     00141          retlw      249
004D 08FF     00142          retlw      255
004E 08FF     00143          retlw      255
004F 08F9     00144          retlw      249
0050 08EE     00145          retlw      238
0051 08DF     00146          retlw      223
0052 08CB     00147          retlw      203
0053 08B4     00148          retlw      180
0054 089A     00149          retlw      154
0055 0880     00150          retlw      128
0056 0865     00151          retlw      101
0057 084B     00152          retlw       75
0058 0834     00153          retlw       52
0059 0820     00154          retlw       32
005A 0811     00155          retlw       17
005B 0806     00156          retlw        6
005C 0800     00157          retlw        0
005D 0800     00158          retlw        0
005E 0806     00159          retlw        6
005F 0811     00160          retlw       17
0060 0820     00161          retlw       32
0061 0834     00162          retlw       52
0062 084B     00163          retlw       75
0063 0865     00164          retlw      101
0064 087F     00165          retlw      127        ; End of this sine wave
0065          00166 sinerow4                      ; Address for sine wave in row 4
0065 089D     00167          retlw      157
0066 08B9     00168          retlw      185
0067 08D2     00169          retlw      210
0068 08E6     00170          retlw      230
0069 08F5     00171          retlw      245
006A 08FE     00172          retlw      254
006B 08FF     00173          retlw      255
006C 08FA     00174          retlw      250
006D 08EE     00175          retlw      238
006E 08DD     00176          retlw      221
006F 08C6     00177          retlw      198
0070 08AB     00178          retlw      171
0071 088E     00179          retlw      142
0072 0871     00180          retlw      113
0073 0854     00181          retlw       84
0074 0839     00182          retlw       57
0075 0822     00183          retlw       34
```


© 1997 Microchip Technology Inc.

```
0076 0811     00184          retlw       17
0077 0805     00185          retlw        5
0078 0800     00186          retlw        0
0079 0801     00187          retlw        1
007A 080A     00188          retlw       10
007B 0819     00189          retlw       25
007C 082D     00190          retlw       45
007D 0846     00191          retlw       70
007E 0862     00192          retlw       98
007F 087F     00193          retlw      127        ; End of this sine wave
0080          00194 sinecolumna                    ; Address for sine wave in column A
0080 08A5     00195          retlw      165
0081 08C8     00196          retlw      200
0082 08E4     00197          retlw      228
0083 08F7     00198          retlw      247
0084 08FF     00199          retlw      255
0085 08FC     00200          retlw      252
0086 08EE     00201          retlw      238
0087 08D7     00202          retlw      215
0088 08B7     00203          retlw      183
0089 0893     00204          retlw      147
008A 086C     00205          retlw      108
008B 0848     00206          retlw       72
008C 0828     00207          retlw       40
008D 0811     00208          retlw       17
008E 0803     00209          retlw        3
008F 0800     00210          retlw        0
0090 0808     00211          retlw        8
0091 081B     00212          retlw       27
0092 0837     00213          retlw       55
0093 085A     00214          retlw       90
0094 087F     00215          retlw      127         ; End of this sine wave
0095          00216 sinecolumnb                    ; Address for sine wave in column B
0095 08A9     00217          retlw      169
0096 08CE     00218          retlw      206
0097 08EB     00219          retlw      235
0098 08FC     00220          retlw      252
0099 08FF     00221          retlw      255
009A 08F5     00222          retlw      245
009B 08DE     00223          retlw      222
009C 08BC     00224          retlw      188
009D 0895     00225          retlw      149
009E 086A     00226          retlw      106
009F 0843     00227          retlw       67
00A0 0821     00228          retlw       33
00A1 080A     00229          retlw       10
00A2 0800     00230          retlw        0
00A3 0803     00231          retlw        3
00A4 0814     00232          retlw       20
00A5 0831     00233          retlw       49
00A6 0856     00234          retlw       86
00A7 087F     00235          retlw      127          ; End of this sine wave
00A8          00236 sinecolumnc            ; Address for sine wave in column C (double sine wave)
00A8 08AD     00237          retlw      173
00A9 08D4     00238          retlw      212
00AA 08F1     00239          retlw      241
00AB 08FF     00240          retlw      255
00AC 08FC     00241          retlw      252
00AD 08E9     00242          retlw      233
00AE 08C8     00243          retlw      200
00AF 089E     00244          retlw      158
00B0 0870     00245          retlw      112
00B1 0844     00246          retlw       68
00B2 0820     00247          retlw       32
00B3 0808     00248          retlw        8
00B4 0800     00249          retlw        0
```


© 1997 Microchip Technology Inc.

```
00B5 0808      00250          retlw       8
00B6 0820      00251          retlw       32
00B7 0844      00252          retlw       68
00B8 0870      00253          retlw      112
00B9 089E      00254          retlw      158
00BA 08C8      00255          retlw      200
00BB 08E9      00256          retlw      233
00BC 08FC      00257          retlw      252
00BD 08FF      00258          retlw      255
00BE 08F1      00259          retlw      241
00BF 08D4      00260          retlw      212
00C0 08AD      00261          retlw      173
00C1 0880      00262          retlw      128
00C2 0852      00263          retlw       82
00C3 082B      00264          retlw       43
00C4 080E      00265          retlw       14
00C5 0800      00266          retlw        0
00C6 0803      00267          retlw        3
00C7 0816      00268          retlw       22
00C8 0837      00269          retlw       55
00C9 0861      00270          retlw       97
00CA 088F      00271          retlw      143
00CB 08BB      00272          retlw      187
00CC 08DF      00273          retlw      223
00CD 08F7      00274          retlw      247
00CE 08FF      00275          retlw      255
00CF 08F7      00276          retlw      247
00D0 08DF      00277          retlw      223
00D1 08BB      00278          retlw      187
00D2 088F      00279          retlw      143
00D3 0861      00280          retlw       97
00D4 0837      00281          retlw       55
00D5 0816      00282          retlw       22
00D6 0803      00283          retlw        3
00D7 0800      00284          retlw        0
00D8 080E      00285          retlw       14
00D9 082B      00286          retlw       43
00DA 0852      00287          retlw       82
00DB 087F      00288          retlw      127         ; End of this sine wave
               00289 ; ********************************************************************************
               00290 ; * sineaddress                                                                   *
               00291 ; * This subroutine is used to calculate actual address in the sinelookup        *
               00292 ; * table for two DTMF waveforms.  It is only called by setdtmfbase. W           *
               00293 ; * is loaded with the key number and this routine returns the sine               *
               00294 ; * lookup address the sine waves.                                               *
               00295 ; * RAM used:   W                                                                *
               00296 ; *      PROGRAM MEM:    25 Words                                                *
               00297 ; ********************************************************************************
00DC           00298 sineaddress                              ; Lookup table for sine address
00DC 01E2      00299          addwf     PCL,F                 ; Add to PC to jump into table
00DD           00300 keyoffset
00DD 0800      00301 k1       retlw     sinerow1-sineoffset       ; Offset for Row 1 sine wave
00DE 087F      00302          retlw     sinecolumna-sineoffset    ; Offset for Column A sine wave
00DF 0800      00303 k2       retlw     sinerow1-sineoffset       ; Offset for Row 1 sine wave
00E0 0894      00304          retlw     sinecolumnb-sineoffset    ; Offset for Column A sine wave
00E1 0800      00305 k3       retlw     sinerow1-sineoffset       ; Offset for Row 1 sine wave
00E2 08A7      00306          retlw     sinecolumnc-sineoffset    ; Offset for Column A sine wave
00E3 0825      00307 k4       retlw     sinerow2-sineoffset       ; Offset for Row 2 sine wave
00E4 087F      00308          retlw     sinecolumna-sineoffset    ; Offset for Column A sine wave
00E5 0825      00309 k5       retlw     sinerow2-sineoffset       ; Offset for Row 2 sine wave
00E6 0894      00310          retlw     sinecolumnb-sineoffset    ; Offset for Column B sine wave
00E7 0825      00311 k6       retlw     sinerow2-sineoffset       ; Offset for Row 2 sine wave
00E8 08A7      00312          retlw     sinecolumnc-sineoffset    ; Offset for Column B sine wave
00E9 0846      00313 k7       retlw     sinerow3-sineoffset       ; Offset for Row 3 sine wave
00EA 087F      00314          retlw     sinecolumna-sineoffset    ; Offset for Column B sine wave
00EB 0846      00315 k8       retlw     sinerow3-sineoffset       ; Offset for Row 3 sine wave
```


© 1997 Microchip Technology Inc.

```
00EC 0894      00316          retlw     sinecolumnb-sineoffset    ; Offset for Column B sine wave
00ED 0846      00317 k9       retlw     sinerow3-sineoffset       ; Offset for Row 3 sine wave
00EE 08A7      00318          retlw     sinecolumnc-sineoffset    ; Offset for Column C sine wave
00EF 0864      00319 k10      retlw     sinerow4-sineoffset       ; Offset for Row 4 sine wave
00F0 087F      00320          retlw     sinecolumna-sineoffset    ; Offset for Column C sine wave
00F1 0864      00321 k11      retlw     sinerow4-sineoffset       ; Offset for Row 4 sine wave
00F2 0894      00322          retlw     sinecolumnb-sineoffset    ; Offset for Column C sine wave
00F3 0864      00323 k12      retlw     sinerow4-sineoffset       ; Offset for Row 4 sine wave
00F4 08A7      00324          retlw     sinecolumnc-sineoffset    ; Offset for Column C sine wave
  00000000     00325 key1=k1-keyoffset       ; Calculation for sine addr for keypad 1
  00000002     00326 key2=k2-keyoffset       ; Calculation for sine addr for keypad 2
  00000004     00327 key3=k3-keyoffset       ; Calculation for sine addr for keypad 3
  00000006     00328 key4=k4-keyoffset       ; Calculation for sine addr for keypad 4
  00000008     00329 key5=k5-keyoffset       ; Calculation for sine addr for keypad 5
  0000000A     00330 key6=k6-keyoffset       ; Calculation for sine addr for keypad 6
  0000000C     00331 key7=k7-keyoffset       ; Calculation for sine addr for keypad 7
  0000000E     00332 key8=k8-keyoffset       ; Calculation for sine addr for keypad 8
  00000010     00333 key9=k9-keyoffset       ; Calculation for sine addr for keypad 9
  00000012     00334 keystar=k10-keyoffset   ; Calculation for sine addr for keypad *
  00000014     00335 key0=k11-keyoffset      ; Calculation for sine addr for keypad 0
  00000016     00336 keypound=k12-keyoffset  ; Calculation for sine addr for keypad #
               00337 ; *************************************************************************
               00338 ; * senddtmf                                                              *
               00339 ; * This subroutine is used to calculate the offset address for          *
               00340 ; * the two sine waves to be sent and initialize the WAVEABASE and WAVEBBASE*
               00341 ; * file registers.  The key number (key0 - key9 or keystar or keypound) is *
               00342 ; * loaded into W before this routine is called.  Next, the DTMF for that  *
               00343 ; * key is sent to the R2R ladder through portB for 320 mS                 *
               00344 ; * Example:                                                              *
               00345 ; * movlw   key1                                                         *
               00346 ; * call    senddtmf                                                    *
               00347 ; * RAM used:        8 bytes                                            *
               00348 ; * PROGRAM MEM:    37 Words                                            *
               00349 ; *************************************************************************
00F5           00350 senddtmf
00F5 0037      00351          movwf   TEMP             ; Initialize temp with key number
00F6 09DC      00352          call    sineaddress      ; Get the addr for sine wave a for this key
00F7 0030      00353          movwf   WAVEABASE        ; Initialize sine wave A base address
00F8 03B0      00354          swapf   WAVEABASE,F    ; swap WAVEABASE so that Z bit is not affected
00F9 0031      00355          movwf   POINTERA         ; Initialize sine wave A pointer address
00FA 0297      00356          incf    TEMP,W           ; Now we get the second sine wave address...
00FB 09DC      00357          call    sineaddress      ; Get the addr for sine wave b for this key
00FC 0032      00358          movwf   WAVEBBASE        ; Initialize sine wave B base address
00FD 03B2      00359          swapf   WAVEBBASE,F    ; swap WAVEBBASE so that Z bit is not affected
00FE 0033      00360          movwf   POINTERB         ; Initialize sine wave B pointer address
00FF 0C20      00361          movlw   20h              ; Place 32 decimal into W for loop counter
0100 0036      00362          movwf   SINECOUNTH       ; Initialize loop counter to 20
0101           00363 loopsine2cyc
0101 0B02      00364          goto    loopsine   ; Waste two cycles to maintain 35 cycle loop count
0102           00365 loopsine
0102 0211      00366          movf    POINTERA,W       ; Place sine wave address into W
0103 0900      00367          call    sinelookup       ; Lookup first sine wave
0104 0034      00368          movwf   NEXTVALUE        ; Place first sine wave into NEXTVALUE
0105 0F7F      00369          xorlw   ENDSINE          ; Update Z bit
0106 0390      00370          swapf   WAVEABASE,W      ; Restore to beginning of sine wave
0107 0643      00371          btfsc   STATUS,Z         ; Skip if not at end of sine wave
0108 0031      00372          movwf   POINTERA         ; Restore start address if at end of sine
0109 02B1      00373          incf    POINTERA,F       ; Move to the next place in the wave
010A 0213      00374          movf    POINTERB,W       ; Place sine wave address into W
010B 0900      00375          call    sinelookup       ; Lookup second sine wave
010C 01F4      00376          addwf   NEXTVALUE,F      ; Add second sine wave into NEXTVALUE
010D 0F7F      00377          xorlw   ENDSINE          ; Update Z bit
010E 0392      00378          swapf   WAVEBBASE,W      ; Pointer of sine wave beginning -> W
010F 0643      00379          btfsc   STATUS,Z         ; Skip if not at end of sine wave
0110 0033      00380          movwf   POINTERB         ; Restore start address if at end of sine
0111 02B3      00381          incf    POINTERB,F       ; Move to next place in the wave
```


© 1997 Microchip Technology Inc.

```
0112 0314      00382          rrf     NEXTVALUE,W      ; Divide by 2, Place output into PORTB
0113 0026      00383          movwf   PORTB            ; Update PORTB with new R2R value
0114 0B15      00384          goto    waste2cyc        ; Waste (2) cycles for 35 total
0115           00385 waste2cyc
0115 02F5      00386          decfsz  SINECOUNT,F      ; Skip if we are done
0116 0B01      00387          goto    loopsine2cyc     ; Do it again! (add 2 cycles as well)
0117 02F6      00388          decfsz  SINECOUNTH,F     ; Skip if we are done
0118 0B02      00389          goto    loopsine         ; Do it again!
0119 0800      00390          retlw   0                ; Return from sine output
               00391 ; ************************************************************************
               00392 ; * init                                                                 *
               00393 ; * This code is used to initialize the PIC.  PORTB is set to zero and all  *
               00394 ; * pins are set to outputs                                               *
               00395 ; * RAM used:       0 bytes                                             *
               00396 ; * PROGRAM MEM:    3 Words                                             *
               00397 ; ************************************************************************
011A           00398 init
011A 0066      00399          clrf    PORTB            ; Init output latches for port B to 0
011B 0040      00400          clrw                     ; Clear W register
011C 0006      00401          tris    PORTB            ; Set all of PORT B to outputs
               00402 ; ************************************************************************
               00403 ; * testallkeys                                                          *
               00404 ; * This code is used to test all possible keys in the keypad.  First each  *
               00405 ; * key address is loaded and then senddtmf is called.                   *
               00406 ; * RAM used:       0 bytes                                             *
               00407 ; * PROGRAM MEM:    25 Words                                            *
               00408 ; ************************************************************************
011D           00409 testallkeys
011D 0C00      00410          movlw   key1         ; Place key "1" address into W
011E 09F5      00411          call    senddtmf     ; Transmit DTMF tone for this key
011F 0C02      00412          movlw   key2         ; Place key "2" address into W
0120 09F5      00413          call    senddtmf     ; Transmit DTMF tone for this key
0121 0C04      00414          movlw   key3         ; Place key "3" address into W
0122 09F5      00415          call    senddtmf     ; Transmit DTMF tone for this key
0123 0C06      00416          movlw   key4         ; Place key "4" address into W
0124 09F5      00417          call    senddtmf     ; Transmit DTMF tone for this key
0125 0C08      00418          movlw   key5         ; Place key "5" address into W
0126 09F5      00419          call    senddtmf     ; Transmit DTMF tone for this key
0127 0C0A      00420          movlw   key6         ; Place key "6" address into W
0128 09F5      00421          call    senddtmf     ; Transmit DTMF tone for this key
0129 0C0C      00422          movlw   key7         ; Place key "7" address into W
012A 09F5      00423          call    senddtmf     ; Transmit DTMF tone for this key
012B 0C0E      00424          movlw   key8         ; Place key "8" address into W
012C 09F5      00425          call    senddtmf     ; Transmit DTMF tone for this key
012D 0C10      00426          movlw   key9         ; Place key "9" address into W
012E 09F5      00427          call    senddtmf     ; Transmit DTMF tone for this key
012F 0C12      00428          movlw   keystar      ; Place key "*" address into W
0130 09F5      00429          call    senddtmf     ; Transmit DTMF tone for this key
0131 0C14      00430          movlw   key0         ; Place key "0" address into W
0132 09F5      00431          call    senddtmf     ; Transmit DTMF tone for this key
0133 0C16      00432          movlw   keypound     ; Place key "#" address into W
0134 09F5      00433          call    senddtmf     ; Transmit DTMF tone for this key
0135 0B1D      00434          goto    testallkeys  ; jump back and do it again!
0136           00435 resetvector
01FF           00436          ORG     1ffh         ; The RESET vector of a 54 is at 1FFh
01FF 0B1A      00437          goto    init         ; Jump to initialion routine
               00438          END
```


© 1997 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXX----------
01C0 : ---------------- ---------------- ---------------- ---------------X
0FC0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   311
Program Memory Words Free:   201

Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

2

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B:

```
MPASM 01.40.01 Intermediate      SINE.ASM   3-31-1997  10:59:22         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;*******************************************************************
                     00002         TITLE "PWM based sine wave generator"
                     00003         LIST P=16C620, R=DEC
                     00004
                     00005         INCLUDE         <P16C620.INC>
                     00001         LIST
                     00002 ; P16C620.INC  Standard Header File, Version 1.01  Microchip Technology, Inc.
                     00164         LIST
2007 3FB1            00006         __CONFIG        _BODEN_OFF&_CP_OFF&_PWRTE_ON&_WDT_OFF&_XT_OSC
                     00007 ;
                     00008 ;*******************************************************************
                     00009 ;       File:           SINE.ASM
                     00010 ;       Author:         Rob Stein
                     00011 ;       Date:           12/20/95
                     00012 ;       Assembler:      MPASM V01.40
                     00013 ;       Xtal:           20 Mhz
                     00014 ;       Inst Clk:       5 Mhz (200nSec)
                     00015 ;*******************************************************************
                     00016 ;       Description:
                     00017 ;       Outputs a 60 Hz sythesized sine wave (32 step) via a general
                     00018 ;       purpose I/O pin (RB1) into a low pass filter. A software PWM
                     00019 ;       routine is used to create 32 separate sinewave steps.   This
                     00020 ;       software was prototyped with the PICDEM1 board.
                     00021 ;
                     00022 ;       Circuit Diagram:
                     00023 ;
                     00024 ;                   2.7k            2.7k
                     00025 ;       RB1  ___/\  /\  /\_____/\  /\  /\_______ Analog Output
                     00026 ;                 \/  \/    |     \/  \/     |
                     00027 ;                           |                |
                     00028 ;                        ----- 0.1uF      ----- 0.1uF
                     00029 ;                        -----            -----
                     00030 ;                          |                |
                     00031 ;                         GND              GND
                     00032 ;
                     00033 ;       ROM Usage: 98 words
                     00034 ;
                     00035 ;       RAM Usage: 6 bytes
                     00036 ;
                     00037 ;************************ Constant Definition   **********************
                     00038
  01312D00           00039 FXTAL           EQU     .20000000       ; Crystal Frequency
  004C4B40           00040 FINST           EQU     FXTAL/4         ; Instruction Cycle Frequency
  0000003C           00041 FSINE           EQU     .60             ; Sine function frequency
  00000020           00042 STEP#           EQU     .32             ; Number of steps
  00000780           00043 FSTEP           EQU     FSINE * STEP#   ; Step frequency
                     00044
                     00045 ;************************  Register Definition  **********************
                     00046
  00000020           00047 TEMPW           EQU     0x20            ; Temporary interrupt storage for W
  00000021           00048 DELAYCNT1       EQU     0x21            ; Delay routine counter low
  00000022           00049 DELAYCNT2       EQU     0x22            ; Delay routine counter high
  00000023           00050 STEPCOUNT       EQU     0x23            ; Sine step counter
  00000024           00051 OUTLOW          EQU     0x24            ; PWM low cycle load for TMR0
```


© 1997 Microchip Technology Inc.

```
  00000025     00052 OUTHIGH EQU     0x25            ; PWM high cycle load for TMR0
               00053
               00054 ;************************     Bit Definition     **********************
               00055
  00000001     00056 PWM     EQU     0x01            ; RB1 used for PWM output
               00057
               00058 ;*****************************************************************
               00059 ;                                 Reset Vector
               00060 ;*****************************************************************
               00061
0000           00062         org     0x000
0000 2816      00063         goto    Start           ; Begining of Program
               00064
               00065 ;*****************************************************************
               00066 ;                 Interupt Vector and Service Routine
               00067 ;       This interrupt routine is entered via an overflow of TMR0 from
               00068 ;       0xFF to 0x00.  A test of RB1 determines if the next time state
               00069 ;       is a high or low cycle.  The next interrupt will occur based the
               00070 ;       TMR0 reload value (OUTLOW or OUTHIGH).
               00071 ;
               00072 ;       The interrupt routine was designed to use a minimial number of
               00073 ;       instruction cycles.  This was done to maximize the PWM duty cycle
               00074 ;       range (ie. a 5 % to 95 % range is achievable with this ISR).  Note
               00075 ;       that 'swapf' instructions are used to perform register moves without
               00076 ;       effecting the STATUS flags (this saves instruction cycles by
               00077 ;       eliminating the need to temporarily save the STATUS register).
               00078 ;
               00079 ;*****************************************************************
               00080
0004           00081         org     0x004         ; Interrupt vector location
0004           00082 IntVector
0004 00A0      00083         movwf   TEMPW         ; Temporarily save W
0005 1886      00084         btfsc   PORTB,PWM     ; Was this a Low cycle ?
0006 280F      00085         goto    PWMLow        ; No ...
0007           00086 PWMHigh
0007 0E25      00087         swapf   OUTHIGH,W     ; Yes... Load hi-time w/o affecting STATUS flags
0008 1486      00088         bsf     PORTB,PWM
0009 0000      00089         nop                   ; Delay to equalize high/low TMR0 load cycles
000A 0081      00090         movwf   TMR0          ; Load next edge interrupt time
000B 110B      00091         bcf     INTCON,T0IF   ; Clear TMR0 overflow flag
000C 0EA0      00092         swapf   TEMPW,F       ; Swap saved W
000D 0E20      00093         swapf   TEMPW,W       ; Restore W
000E           00094 IntEndHi
000E 0009      00095         retfie                ; Return from Interrupt
000F           00096 PWMLow
000F 1086      00097         bcf     PORTB,PWM
0010 0E24      00098         swapf   OUTLOW,W      ; Load low time
0011 0081      00099         movwf   TMR0          ; Load next edge interrupt time
0012 110B      00100         bcf     INTCON,T0IF   ; Clear TMR0 overflow flag
0013 0EA0      00101         swapf   TEMPW,F       ; Swap saved W
0014 0E20      00102         swapf   TEMPW,W       ; Restore W
0015           00103 IntEndLo
0015 0009      00104         retfie                ; Return from Interrupt
               00105
               00106 ;*****************************************************************************
               00107 ;                                 Main Routine
               00108 ;*****************************************************************************
0016           00109 Start
0016 0183      00110         clrf    STATUS        ; Intitialize STATUS & select bank 0
0017 1683      00111         bsf     STATUS,RP0    ; Select register bank 1
0018 3088      00112         movlw   0x88
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0019 0081      00113         movwf   OPTION_REG    ; 1:1 TMR0 prescaler, PORTB pull-ups disabled
001A 30FF      00114         movlw   0xFF
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
001B 0085      00115         movwf   TRISA         ; Set Port_A as inputs
```

```
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
001C 0186      00116          clrf    TRISB           ; Set Port_B as outputs
001D 1283      00117          bcf     STATUS,RP0      ; Select register bank 0
001E 0086      00118          movwf   PORTB           ; PORT_B pins high
001F 0181      00119          clrf    TMR0            ; Initialize TMR0
0020 30A0      00120          movlw   0xA0
0021 008B      00121          movwf   INTCON          ; Enable TMR0 and global interrupt
0022           00122 ResetStep
0022 3020      00123          movlw   STEP#
0023 00A3      00124          movwf   STEPCOUNT       ; Load counter for 32 steps
0024           00125 StepLoop
0024 2059      00126          call    Delay           ; Software delay
0025 0823      00127          movf    STEPCOUNT,W     ; Pass table offset via W
0026 2037      00128          call    SineTable       ; Get table value
0027 202B      00129          call    SetPWM          ; Set-up low & high PWM values
0028 0BA3      00130          decfsz  STEPCOUNT,F     ; Next step
0029 2824      00131          goto    StepLoop
002A 2822      00132          goto    ResetStep
               00133
               00134 ;*************************************************************************
               00135 ;                                 Set PWM Subroutine                     *
               00136 ;       The following calculates the next low and high PWM time values.  *
               00137 ;       The two time values, OUTLOW and OUTHIGH, will be passed to the    *
               00138 ;       interrupt service routine.                                       *
               00139 ;*************************************************************************
002B           00140 SetPWM
002B 138B      00141          bcf     INTCON,GIE      ; Disable interrupts to protect ISR from...
               00142                                  ; corrupting OUTLOW & OUTHIGH values
002C 00A4      00143          movwf   OUTLOW          ; Set PWM Duty Cycle
002D 0924      00144          comf    OUTLOW,W
               00145
002E 3E0A      00146          addlw   IntEndHi-IntVector ; Adjust for Int Service time
002F 00A5      00147          movwf   OUTHIGH
0030 0824      00148          movf    OUTLOW,W
0031 3E0A      00149          addlw   IntEndHi-IntVector ; Adjust for Int Service time
0032 00A4      00150          movwf   OUTLOW
               00151
0033 0EA4      00152          swapf   OUTLOW,F        ; Swap nibbles so that interrupt service...
0034 0EA5      00153          swapf   OUTHIGH,F       ; will not corrupt STATUS
0035 178B      00154          bsf     INTCON,GIE      ; Re-enable interrupts
0036 0008      00155          return
               00156
               00157 ;*************************************************************************
               00158 ;                        Lookup Table for Sine Wave
               00159 ;       This 32 entry table was generated to produce a 0.1*Vdd to
               00160 ;       0.9*Vdd (typicaly 0.5 to 4.5 volt) sine function.
               00161 ;*************************************************************************
0037           00162 SineTable
0037 0782      00163          addwf   PCL,F           ; Increment into table
0038 3400      00164          retlw   .0              ; Dummy table value
0039 3480      00165          retlw   .128            ; 0 degree, 2.5 volt
003A 3494      00166          retlw   .148
003B 34A7      00167          retlw   .167
003C 34B9      00168          retlw   .185
003D 34C8      00169          retlw   .200
003E 34D5      00170          retlw   .213
003F 34DE      00171          retlw   .222
0040 34E4      00172          retlw   .228
0041 34E6      00173          retlw   .230            ; 90 degree, 4.5 volt
0042 34E4      00174          retlw   .228
0043 34DE      00175          retlw   .222
0044 34D5      00176          retlw   .213
0045 34C8      00177          retlw   .200
0046 34B9      00178          retlw   .185
0047 34A7      00179          retlw   .167
```


© 1997 Microchip Technology Inc.

```
0048 3494     00180         retlw   .148
0049 3480     00181         retlw   .128           ; 180 degree, 2.5 volt
004A 346C     00182         retlw   .108
004B 3459     00183         retlw   .89
004C 3447     00184         retlw   .71
004D 3438     00185         retlw   .56
004E 342B     00186         retlw   .43
004F 3422     00187         retlw   .34
0050 341C     00188         retlw   .28
0051 341A     00189         retlw   .26           ; 270 degree, 0.5 volt
0052 341C     00190         retlw   .28
0053 3422     00191         retlw   .34
0054 342B     00192         retlw   .43
0055 3438     00193         retlw   .56
0056 3447     00194         retlw   .71
0057 3459     00195         retlw   .89
0058 346C     00196         retlw   .108
              00197
              00198 ;*******************************************************************************
              00199 ;                       Time Delay Sub-routine                                 *
              00200 ;       The time delay is used to create the precision 32 steps.  The          *
              00201 ;       32 step times totaled together add up to a 60 Hz rate. Note that       *
              00202 ;       constants DELAYCNT# are used so that other frequencies can easily      *
              00203 ;       be generated (example: FSINE equ .50 for a 50 Hz  sinewave).           *
              00204 ;*******************************************************************************
  00000A2C    00205 TDELAY          EQU     FINST/FSTEP     ; # of delay count cycles
  0000032D    00206 ADJTDELAY       EQU     TDELAY/3 - 55   ; Adjust for main routine cycles
  00000003    00207 TDELAYHI        EQU     high ADJTDELAY  ; Most Significant Byte of TDELAY
  0000002D    00208 TDELAYLO        EQU     low ADJTDELAY   ; Least Sig. Byte of TDELAY
              00209
0059          00210 Delay
0059 3003     00211         movlw   TDELAYHI
005A 00A2     00212         movwf   DELAYCNT2               ; Load high byte delay counter
005B 01A1     00213         clrf    DELAYCNT1
005C          00214 LoopD1
005C 0BA1     00215         decfsz  DELAYCNT1,F     ; Finished with 256 loops ?
005D 285C     00216         goto    LoopD1          ; No ... keep going
005E 0BA2     00217         decfsz  DELAYCNT2,F     ; Yes... Done with TDELAYHI loops ?
005F 285C     00218         goto    LoopD1          ; No ...
              00219
0060 302D     00220         movlw   TDELAYLO        ; Yes... Load low byte with adjust for...
0061 00A1     00221         movwf   DELAYCNT1       ; main routine cycles.
0062          00222 LoopD2
0062 0BA1     00223         decfsz  DELAYCNT1,F     ; Finished with TDELAYLO loops ?
0063 2862     00224         goto    LoopD2          ; No ... keep going
0064 0008     00225         return                  ; Yes... Finished
              00226
              00227         END                     ; That's all Folks !

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)


0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXX----------- ----------------
2000 : -------X-------- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:    98
Program Memory Words Free:   414


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     3 reported,     0 suppressed
```

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN657

Decoding Infrared Remote Controls Using a PIC16C5X Microcontroller

Author: William G. Grimm
Consultant

INTRODUCTION

For many years the consumer electronics industry has been employing infrared remote controls for the control of televisions, VCR's, and cable boxes. This same technology has recently started to appear in industrial applications to eliminate keypads.

Decoding most of the infrared signals can be easily handled by PIC16C5X microcontrollers. This application note describes how this decoding may be done.

The only mandatory hardware for decoding IR signals is an infrared receiver. The use of two types is described here. Both are modular types used often by the consumer electronics industry. The first type responds to infrared signals modulated at about 40 kHz. The second responds to non-modulated infrared pulses and has a restricted range. The hardware costs of each approach will be less than two dollars.

Three PIC16C5X application programs are described, and instructions on how they can be used to create an algorithm that can decode just about any remote control signal. Each PIC16C5X application program represents a step in mapping out a pre-existing infrared format. The final application is a fully implemented example of decoding and interpreting the infrared signals of a type of Teknika TV remote.

THE THREE LAYERS OF AN INFRARED SIGNAL

The typical infrared signal used by remote controls has three layers. The names used for these layers has not been standardized. In this application note they are called the infrared, the modulation, and the serial data.

The infrared layer is the means of transmission. Infrared is light whose wavelength is too long to see. Although you cannot see the infrared beam, it behaves the same as light, so if you cannot see the target device, you cannot control it with an infrared signal. To control around corners or through opaque materials, RF, usually UHF signals are used. Although this application note does not further mention RF, much of what is presented here can be used with an RF transmission medium.

The modulation layer refers to the fact that each burst of infrared signal is often modulated at a frequency between 32.75 kHz and 56.8 kHz. This is done to diminish the effects of ambient light. This layer, however, is optional. Some infrared formats do not modulate their outputs, sending pulses of unmodulated infrared light instead. This is done to extend the remote control's battery life and to reduce the cost of the remote control device.

The serial data layer has the information containing a command. This is typically coded in the lengths of infrared bursts or in the lengths of gaps between infrared bursts. A long gap or burst is interpreted as a '1', a short gap or burst is interpreted as a '0'.

HARDWARE DESCRIPTION

The schematic in Figure 1 shows a tool that can be made to aid development of infrared receiver code. The schematic consists of a PIC16C57 connected to one of two available infrared receivers. One receiver is for non-modulated signals, the other for modulated signals. Modulated receivers are available from Sharp and LiteOn, part numbers GP1U521Y and LT-1060 respectively. The non-modulated type is available from Quality Technologies part number QSE157QT.

The choice of the PIC16C57 is not indicative of the processing power required for decoding. Typical IR receiver code can fit into less than half the ROM space available in a PIC16C54, and uses four RAM locations. The choice of a PIC16C57 in this case was driven by the need to store a lot of signal lengths for later reading.

A ceramic resonator clocks the PIC16C57. It will give adequate frequency accuracy to determine pulse and gap lengths. A RC network does not usually have adequate accuracy.

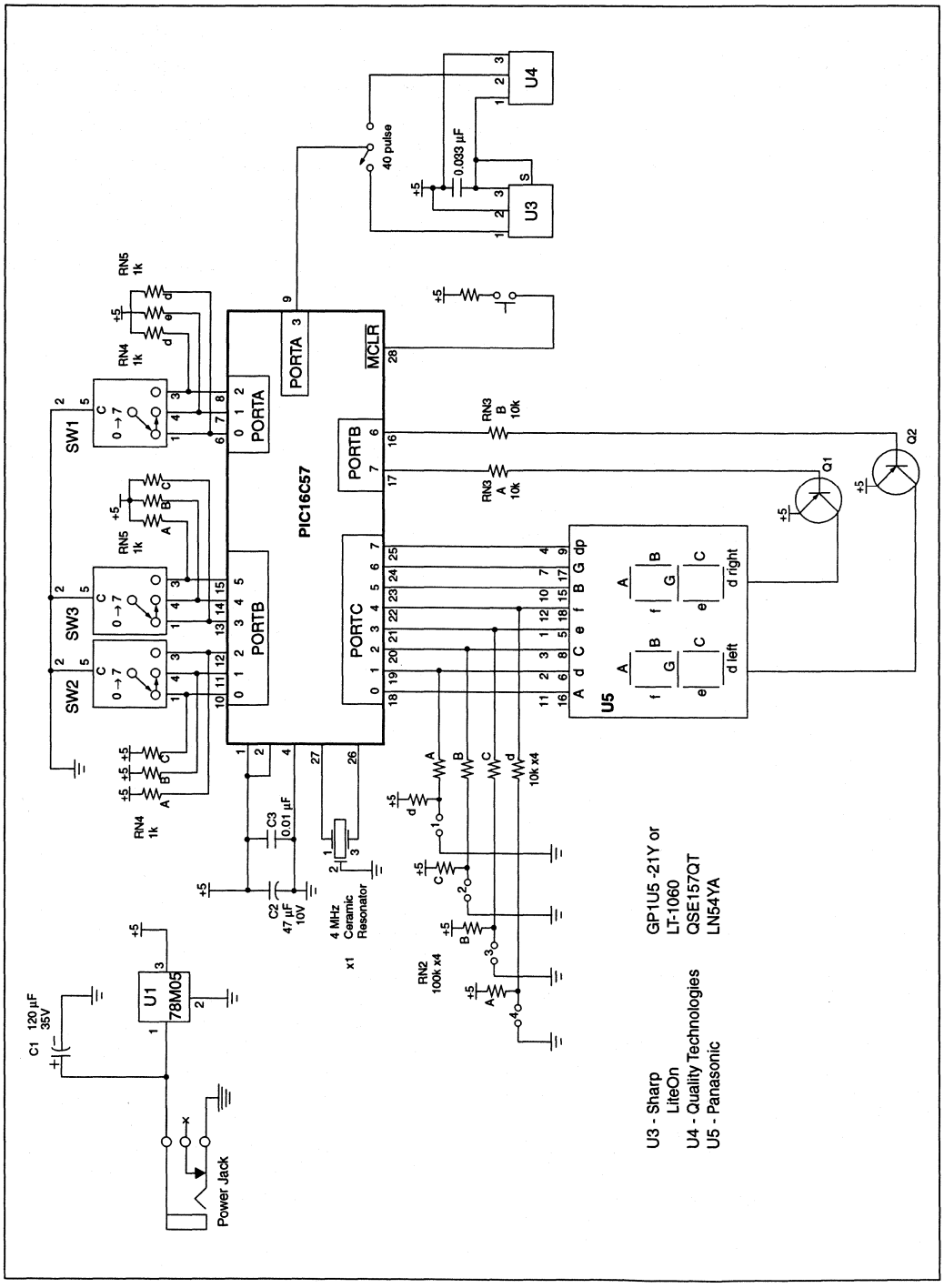
A button is available for resetting the PIC16C57, and four jumpers are provided to control the application start-up. The two digit display is multiplexed and driven through Q1 and Q2.

Three octal switches are used as inputs to control the OPTION register and which file is displayed.

The whole circuit derives its power from a 9V, 200 mA wall mounted supply. U1 regulates the 9V down to 5V for the PIC16C57 and associated circuitry.

© 1997 Microchip Technology Inc.

FIGURE 1: IR RECEIVER SCHEMATIC



© 1997 Microchip Technology Inc.

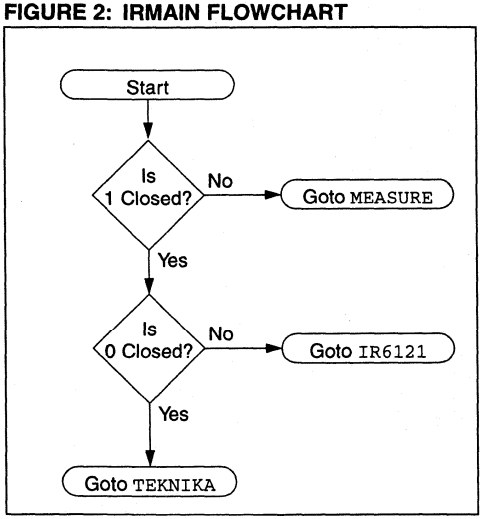
DESCRIPTION OF SOFTWARE TO AID DEVELOPMENT

This application uses four different firmware files. IRMAIN.ASM controls the selection of the three application files. The first file is MEASURE.ASM which stores the infrared burst and gap lengths into memory and allows playback of that information. IR6121.ASM decodes NEC6121 infrared format and displays the received codes on the LED display. The final file, TECNICKA.ASM, shows the final firmware for decoding the infrared format for a Teknika Television.

IRMAIN.ASM

The firmware listed includes three applications that will aid in designing an infrared control system. IRMAIN.ASM reads jumpers 1 and 2 and directs program flow after reset to one of the three applications. Having no jumper in 2 will direct program flow to MEASURE.ASM. A jumper in 2 only will direct program flow to IR6121.ASM. Jumpers in both 1 and 2 will direct program flow to TEKNIKA.ASM. Jumpers 3 and 4 are not used.

FIGURE 2: IRMAIN FLOWCHART



MEASURE.ASM

This is the most basic and most useful of the three applications. This program stores the infrared burst and gap lengths into memory, allowing playback of the measurements through the two digit display. It allows external control of the OPTION register also, through SW1. The setting of SW1 is read directly into the OPTION register prescaler value for TMR0. If SW1 is changed during program operation, the PIC16C57 resets.

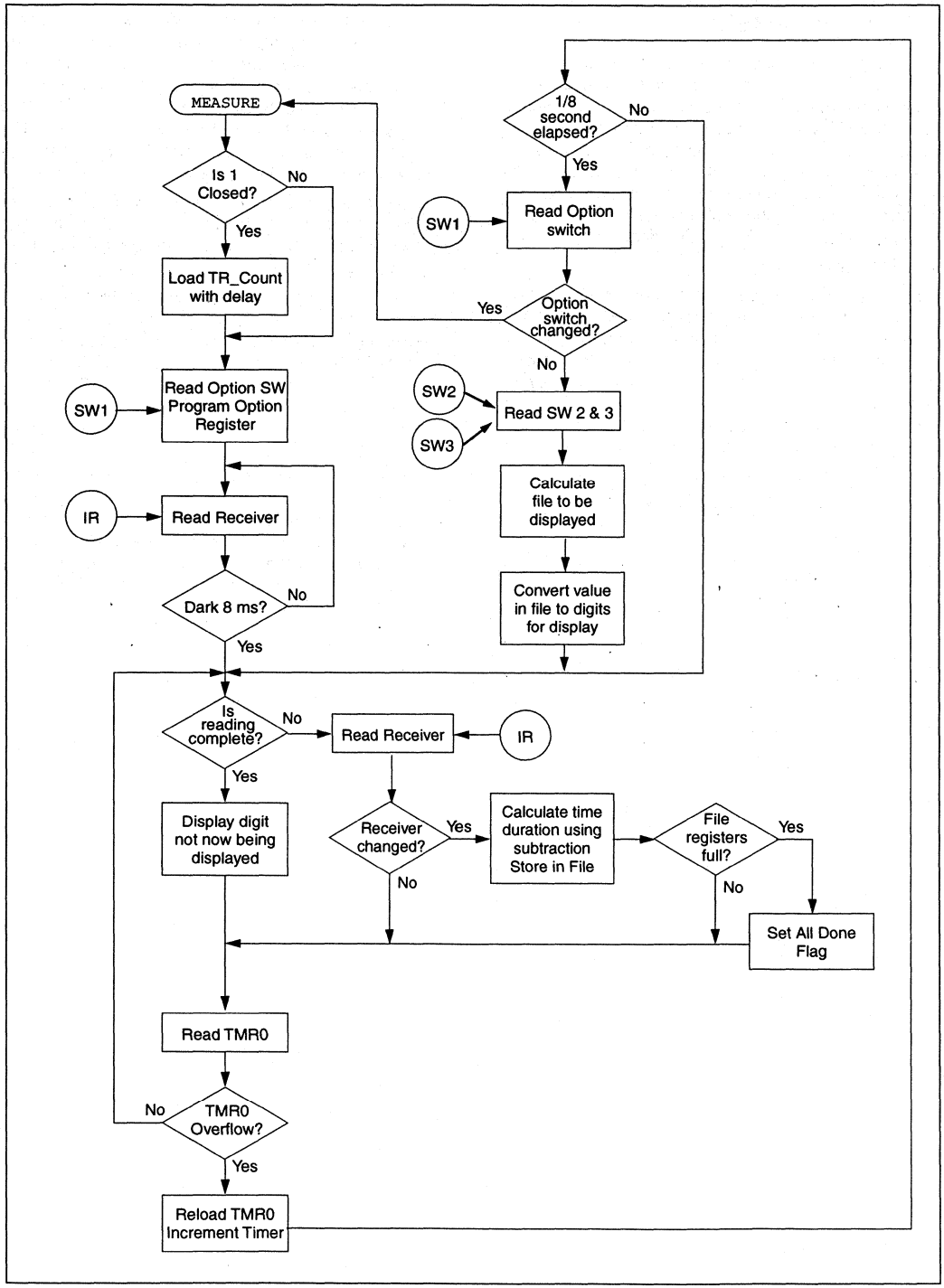
Upon start-up a "hyphen" will be displayed in the left digit space until the infrared input settles to the dark logic indicating that the unit is ready to receive an infrared signal.

As an infrared signal comes in, the lengths of bursts of infrared, and the lengths of gaps between burst are stored in consecutive file locations until all four pages of the PIC16C57's memory files are filled. If a jumper had been in 1, the program throws away the first 32 pulse and gap lengths and starts storing pulse and gap lengths with the thirty third pulse length. This allows the decoding of very long formats.

When all four pages of file memory are filled with pulse and gap lengths, a number and decimal point are displayed. The decimal point indicates that the unit is done reading. The number is a gap or pulse length. SW2 and SW3 control the time sequence of the pulse or gap length displayed. These are in octal with SW3 being the more significant digit.

© 1997 Microchip Technology Inc.

FIGURE 3: MEASURE.ASM FLOWCHART

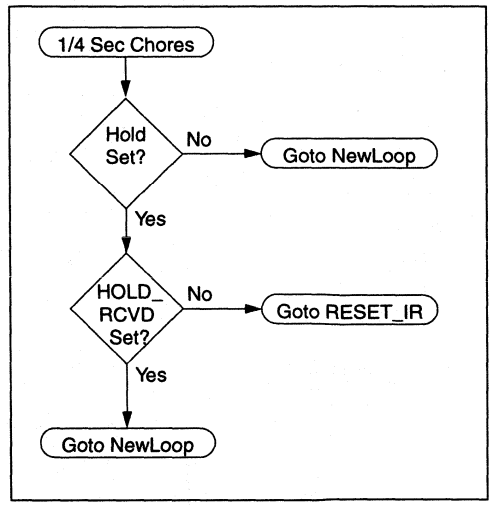


© 1997 Microchip Technology Inc.

IR6121.ASM

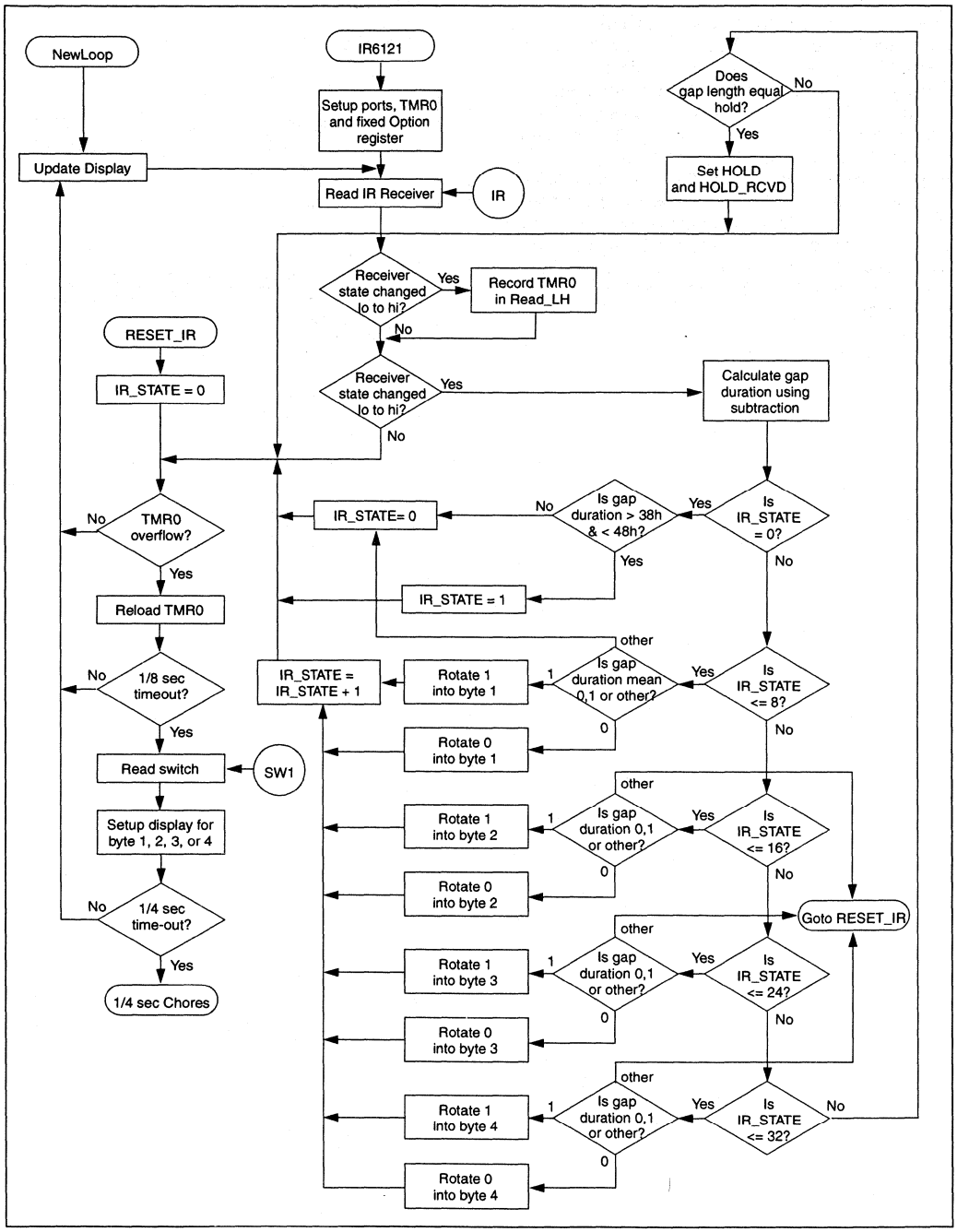
This is an example of the next stage in development. It uses the IR receiver, PIC16C57 clock frequency, OPTION prescaler value and characteristic time length constants that were found after using MEASURE.ASM with an infrared remote control based on the NEC6121 infrared controller[1]. The resulting algorithm is able to decode the infrared bit stream and display it as four bytes on the two digit display. The bytes are switched using SW1 (changing it will not cause this application to reset). From it, or such a program customized to your particular remote control, a list can be made of how each button on a remote control resolves to a set of bytes in memory. This allows the creation of a button lookup table.

FIGURE 4: 1/4 SECOND CHORES



© 1997 Microchip Technology Inc.

FIGURE 5: `IR6121.ASM` FLOWCHART



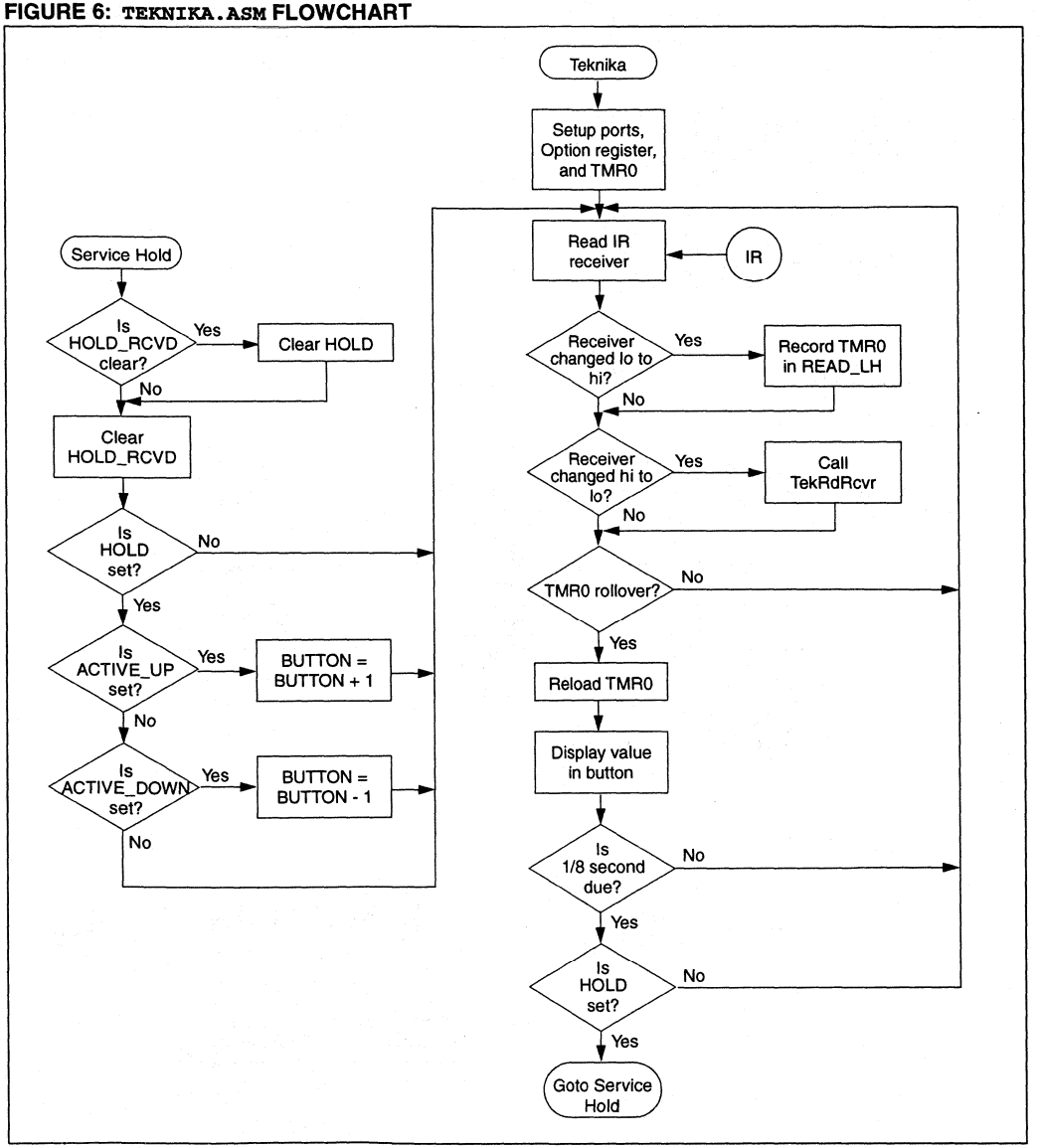
© 1997 Microchip Technology Inc.

TEKNIKA.ASM

This is an example of a finished product. This program fully decodes the infrared format of a Teknika Television. When a number is pressed on the remote control it is displayed on the display. When channel up or channel down is pressed, the displayed number increases or decreases.

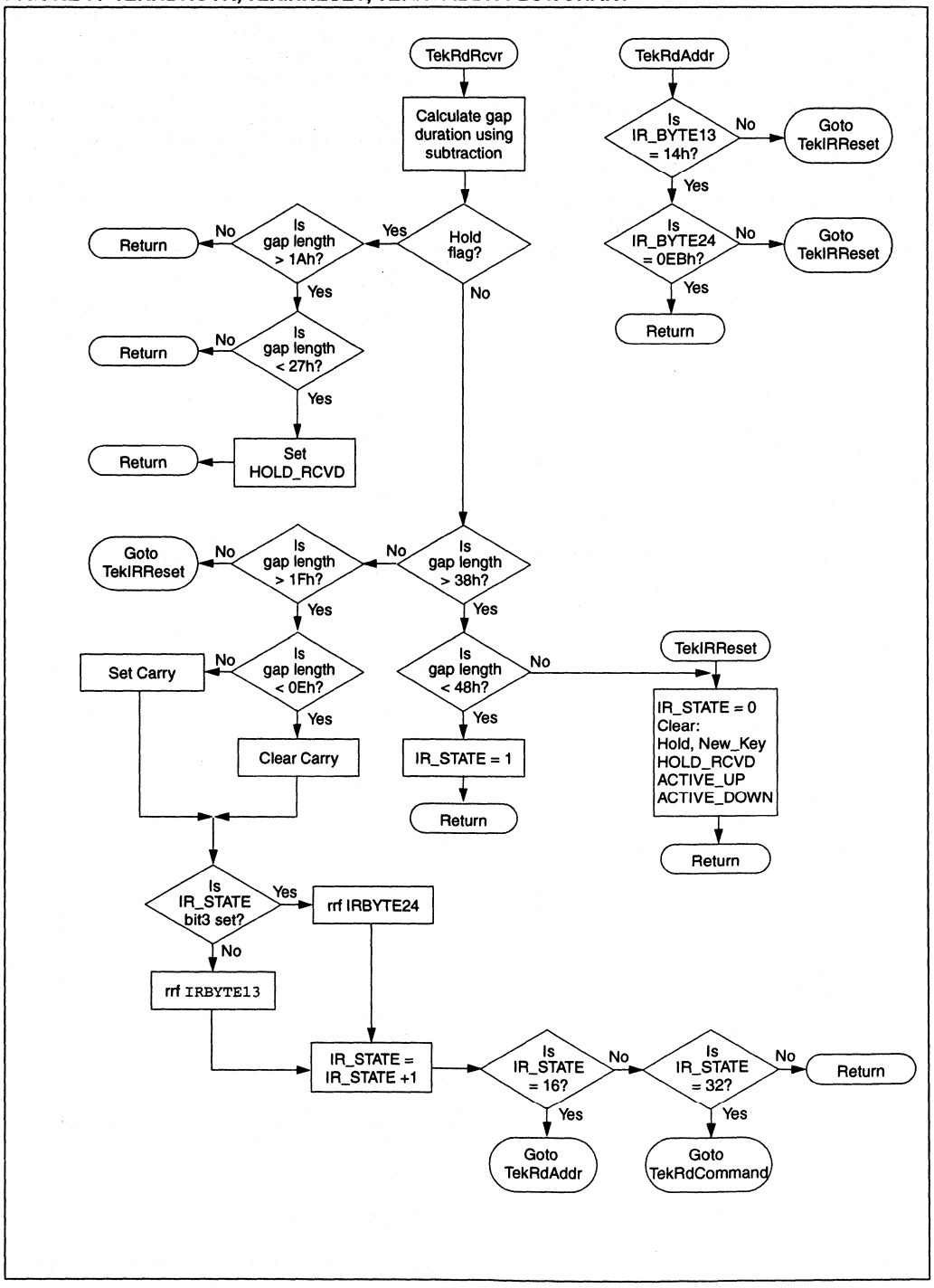
It incorporates the final step in implementing a remote control decoder, that of cross referencing codes to button numbers. The algorithm will only respond if the first two bytes are 14h and EBh, the characteristic of this type of Teknika television. Byte 3 and byte 4 are checked to see if they are complements. If so, byte 3 is sent through a lookup table to determine which button the received byte corresponds to, then the appropriate action is taken. The lookup table was made by using IR6121 and recording byte 3 with the button pressed. Similar tables can be made using other remote controls.

FIGURE 6: TEKNIKA.ASM FLOWCHART



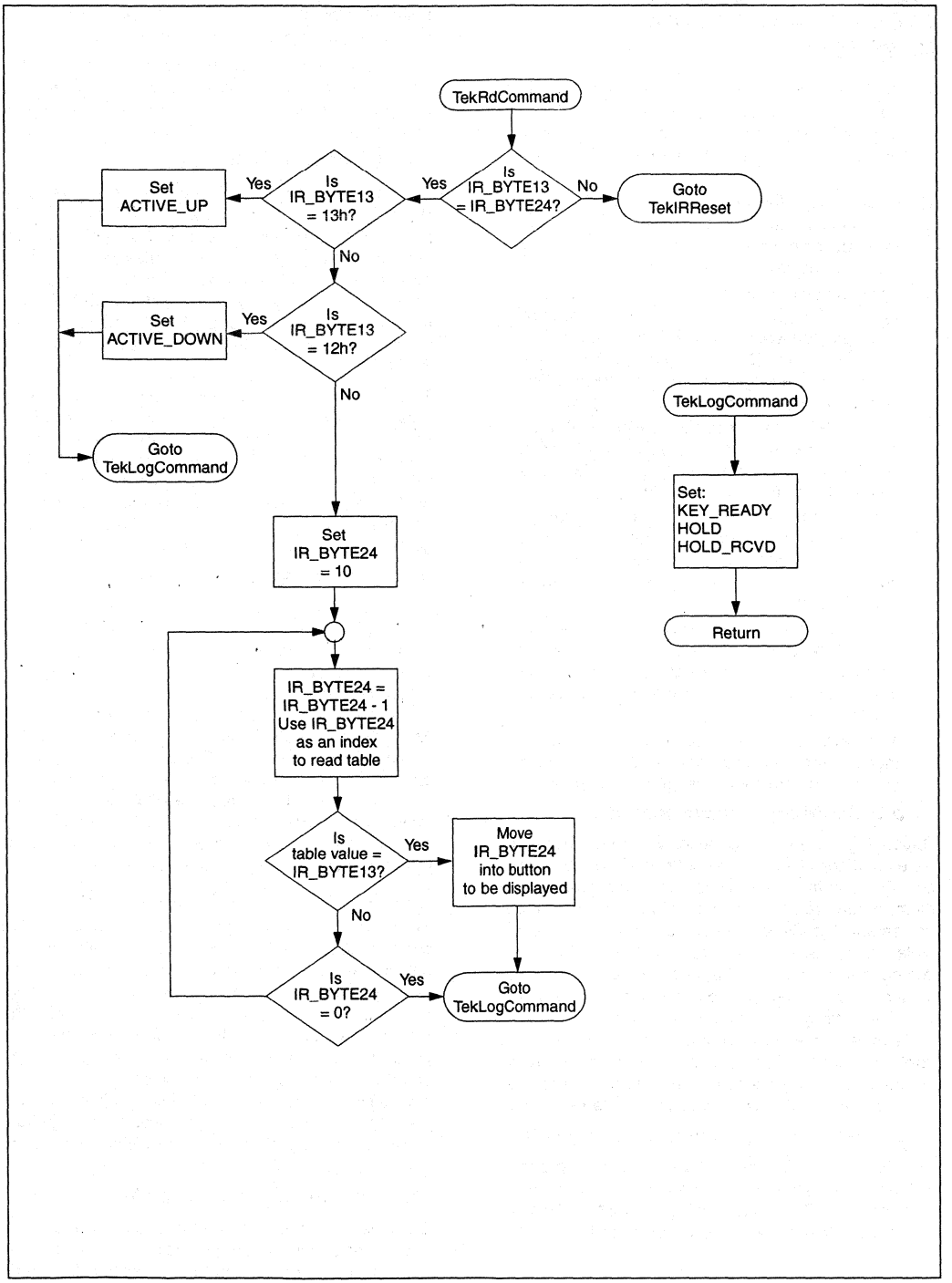
© 1997 Microchip Technology Inc.

FIGURE 7: TEKRDRCVR, TEKIRRESET, TEKRDADDR FLOWCHART



© 1997 Microchip Technology Inc.

FIGURE 8: TEKCOMMAND FLOWCHART



© 1997 Microchip Technology Inc.

INSTRUCTIONS ON WRITING AN ALGORITHM TO DECODE IR REMOTES

1. To design a system that uses an infrared remote control, the first step is to choose a remote control. Self designed or off the shelf, modulated or unmodulated are the primary technical decisions.
2. Once a remote control has been chosen or designed, its modulation frequency, if it has one, must be determined. This controls the kind of hardware used to receive the infrared signal.
3. The next step is to determine the time-base of the data, that is, if the pulses and gaps are short or long in reference to the PIC16C57 clock. The OPTION switch, SW1, is used to get optimum length pulse and gap counts from TMR0. This defines the value of the OPTION prescaler.
4. Fourth, definition is made as to what, in the format, defines a '1', and what, in the format, defines a '0'. This could be gap counts, pulse counts, or a combination of both.
5. Fifth, determination is made of the full length of commands. This enables the determination as to whether a button is being held down or if a new command of the same type as previous is being issued.
6. The sixth step requires the writing of code. The code will resolve the gap and pulse lengths and command lengths into bits and bytes. Each button on the remote will decode to a unique series of bits.
7. The seventh and final step takes these codes that are received and converts them to button numbers or commands, using a lookup table.

Step 1: Choosing a remote control

Depending on your application, you may choose to have an off the shelf remote control or design one yourself. Typically they have small 4-bit microcontrollers in them, preprogrammed for a serial format. Some companies such as General Instrument sell them as complete units, others such as NEC sell the main component which can be customized by external diodes to not interfere with other applications. It is also possible to program a PIC16C57 to generate a signal that can be sent to an infrared LED for transmission. Yet another approach is to use a programmable remote control to generate any number of infrared formats and use them right off the shelf to control the target device.

Step 2: Determining a modulation frequency

For this and the next step the MEASURE.ASM program will be used. To start out, use the non-modulated receiver and a PIC16C57 running the MEASURE.ASM application. Select 1 on the option selector. Press a button until the decimal point comes on. Using the jumpers switch through the memorized pulse durations that the PIC16C57 will have stored in its memory.

If all of the reading except the first are below 40h, the infrared format is a modulating one. If half or more of the values show up as 0FFh, then the remote is non-modulating.

Step 3: Determine time-base

If the remote control is modulated, switch to a demodulating IR receiver. With the option selector still at 1, press a button on the remote control again until the decimal point comes on. The series of memorized pulse durations will now probably include a lot of FFh values. If so, move the Option selector up until the values are in the 7h to 1Fh range. The Option selector has the optimum value for the option divisor to be used in the TMR0 register.

To optimize range and reliability, several demodulating receivers may be tried. These are available from Sharp or LiteOn. The modulating frequencies that are presently used are 32.75 kHz, 35.0 kHz, 36.0 kHz, 36.7 kHz, 38 kHz, 39 kHz, 40 kHz, 41.7 kHz, 48 kHz, and 56.8 kHz[2]. The most common are round 40 kHz. The best match for your remote control will give the longest range and most consistent results.

Step 4: Decoding ones and zeros

The next step is to map out the characteristic pulse and gap lengths that represent ones and zeros. By pressing the same button on the remote, write down the series of numbers read by the PIC16C57 running the `MEASURE.ASM` program. Each odd numbered entry is the duration of a burst of IR from the remote control. Each even numbered entry is the duration of a gap between bursts of infrared. The lengths of these gaps and bursts define ones and zeros. Their order will depend on which button is pressed. Once the characteristic lengths have been discovered for a one and a zero, an algorithm can then be created with a counter to translate the lengths into ones and zeros.

Step 5: Finding the Command Length

Press the same button again. The command duration can also be found. This is necessary to determine if a button is being held down or a new command of the same type is being issued. Most remote controls repeat the command as long as the button is held down, the repetitions separated by a long dark time, usually 0FFh on an even numbered transition. If no long even numbered counts can be found, consider that some commands can be longer than 64 transitions. The option to delay counting is available for this reason. Insert jumper 1 and `MEASURE.ASM` will only start storing transition times after the 32nd transition.

Step 6: Translating lengths to bits

Once the characteristic lengths of ones and zeros have been found and the length of the typical command has been found, a program can then be written to decode

© 1997 Microchip Technology Inc.

these lengths to ones and zeros and display them on the two digit display. Also a HOLD flag can be created which will be true as long as the button is being held down. Usually 1/8 second between commands indicates a new command. Use this value to time out HOLD times and times between commands. IR6121.ASM is an example of a program that translates the gap lengths of the NEC6121 format to the four bytes that make up the information in each command.

Step 7: Create a button to code cross reference table

TEKNIKA.ASM implements a lookup table to translate the codes received to the actual button pressed. A counter is loaded with the highest number button that can be pressed, and the code is then looked up and compared with the code that was received. If no match, the counter is decremented until a match is found. When found, the counter then has the button number that was pressed.

Note too, that more checking may be done at this level on some formats, such as having an address, a complement of the code following the code itself for checking.

The result from all of the steps in decoding is that if a button is pressed on a remote control, that button number appears in a PIC16C57 file location. A command such as channel up or channel down will appear as two set flags, one to indicate the command, the other to indicate that it is active, HOLD. From this point the application can access these flags and files to respond appropriately.

REFERENCES

[1] Infrared Remote Controls ICs; NEC Electronics. August 1991. Literature available 1-800-632-3531.

[2] Sharp Optoelectronics Data Book, 1991/1992, page 961.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:.

```
MPASM 01.40 Released           IRMAIN.ASM   10-2-1996  10:24:32         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

          00001 TITLE        "IR Receiver Ap-Note Program Selector V0.02"
          00002 ;
          00003 ;*******************************************************************
          00004 ; File Name :      IRMAIN.ASM
          00005 ;*******************************************************************
          00006 ;      Author:    William G. Grimm
          00007 ;      Company:   Microchip Technology
          00008 ;      Revision:  V0.02
          00009 ;      Date:      March 28, 1996
          00010 ;      Assembler: MPASM version 1.21
          00011 ;
          00012 ;*******************************************************************
          00013 ;   Revision History:
          00014 ;
          00015 ;
          00016 ;   V0.01    Original  March 28, 1996
          00017 ;
          00018 ;   V0.02    Added the Technika routine March 28, 1996
          00019 ;
          00020 ;
          00021 ;*******************************************************************
          00022 ; Assembly options
          00023           LIST     P=16C57,r=hex,N=75,C=120,T=ON
          00024 ;
          00025 ;*******************************************************************
          00026 ;   Required modules:
          00027 ;
          00028 ;       picreg.equ
          00029 ;       irundef.asm           ; undefines constants for include files
          00030 ;       measure.asm           ; decodes all remotes
          00031 ;       ir6121.asm            ; decodes all 6121 format remotes
          00032         include     "p16c5x.inc"
          00001         LIST
          00002 ; P16C5X.INC  Standard Header File, Version 3.30 Microchip Tech, Inc.
          00224         LIST
          00033 ;
          00034 ;*******************************************************************
          00035 ;
          00036 ;
          00037 ;
          00038 ;*******************************************************************
          00039 ; Ceramic resonator or Crystal, 4.000Mhz. If a Prescaler of
          00040 ; 64 (5 in the option register) is used, TMR0 will increment every
          00041 ; 64.00 microseconds.
          00042 ; If TMR0 is initially loaded with 131, TMR0 will overflow to 0
          00043 ; in 8.000 milliseconds.
00000083  00044 MSEC8          EQU    D'131'
          00045 ;*******************************************************************
          00046 ;
          00047 ;
          00048 ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
          00049         SUBTITL "File and Flag definitions."
          00050 ;*******************************************************************
          00051 ; file memory location definitions
```


© 1997 Microchip Technology Inc.

```
                00052 ;*******************************************************************************
                00053 ;
                00054 ; full byte file memory locations
                00055 ;
0000000E        00056 LEFT_DIGIT    EQU   0E        ; code to be displayed on left digit
0000000F        00057 RIGHT_DIGIT   EQU   0F     ; code to be displayed on right digit
                00058
                00059 ;
                00060 ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                00061         SUBTITL "Constant definitions."
                00062 ; ***************************************************************************
                00063 ; Definition of program constants
                00064         SUBTITL "Constant definitions."
                00065 ; ***************************************************************************
                00066 ; Definitions of constants used to configure the ports
0000000F        00067 A_CONFIG  EQU  0f          ; IR input and option dial all inputs
                00068 ;
0000003F        00069 B_CONFIG  EQU  03f         ; PORTB has outputs on bits 6 and 7 for
                00070                            ; the display, bits 0 to 5 are inputs for
                00071                            ; the dials
                00072 ;
0000001E        00073 C_CONFIG1 EQU  01e         ; lines 1,2,3, and 4 are input during selection
00000000        00074 C_CONFIG2 EQU  0           ; all lines are outputs normally
                00075 ;
                00076 ;*******************************************************************************
                00077 ; Definition of program constants
                00078 ;
000000BF        00079 HIPHEN   EQU B'10111111'          ; -  for display
                00080 ;
                00081 ;
                00082 ;DEFINE PORT_A REG FUNCTION:
                00083 ;       BIT # 7|6|5|4|3|2|1|0|
                00084 ;--------------|-|-|-|-|-|-|-|
                00085 ;              | | | | | | |Y| --> OPTION dial bit 0
                00086 ;              | | | | | |Y| | --> OPTION dial bit 1
                00087 ;              | | | | |Y| | | --> OPTION dial bit 2
                00088 ;              | | | |Y| | | | --> IR input
                00089 ;              | | |O| | | | | --> Unavailable
                00090 ;              | |O| | | | | | --> Unavailable
                00091 ;              |O| | | | | | | --> Unavailable
                00092 ;             O| | | | | | | | --> Unavailable
                00093 ;
00000003        00094 IR            EQU  3              ; IR  receiver
                00095 ;
                00096 ;DEFINE PORT_B REG FUNCTION:
                00097 ;       BIT # 7|6|5|4|3|2|1|0|
                00098 ;--------------|-|-|-|-|-|-|-|
                00099 ;              | | | | | | |Y| --> Right dial bit 0
                00100 ;              | | | | | |Y| | --> Right dial bit 1
                00101 ;              | | | | |Y| | | --> Right dial bit 2
                00102 ;              | | | |Y| | | | --> Left dial bit 0
                00103 ;              | | |Y| | | | | --> Left dial bit 1
                00104 ;              | |Y| | | | | | --> Left dial bit 2
                00105 ;              |Y| | | | | | | --> controls right digit, LOW is on
                00106 ;             Y| | | | | | | | --> controls left digit, LOW is on
                00107 ; Y = DEFINED AS SHOWN (0/1)
                00108 ;
00000006        00109 RIGHT_OFF     EQU  6
00000007        00110 LEFT_OFF      EQU  7
                00111 ;
                00112 ;DEFINE PORT_C REG FUNCTION:
                00113 ;       BIT # 7|6|5|4|3|2|1|0|
                00114 ;--------------|-|-|-|-|-|-|-|     0 turns element on
                00115 ;              | | | | | | |Y| --> A
                00116 ;              | | | | | |Y| | --> D  SW1
                00117 ;              | | | | |Y| | | --> C  SW2
```

© 1997 Microchip Technology Inc.

```
                        00118 ;              | | | |Y| | | | --> E
                        00119 ;              | | |Y| | | | | --> F
                        00120 ;              | |Y| | | | | | --> B
                        00121 ;              |Y| | | | | | | --> G
                        00122 ;             Y| | | | | | | | --> dp
                        00123 ; Y = DEFINED AS SHOWN (0/1)
                        00124 ;
00000001                00125 SW1            EQU  1
00000002                00126 SW2            EQU  2
00000007                00127 DP             EQU  7
                        00128 ;
                        00129 ;^^^
```

```
                00184 ;  Turns off the display at the three transistors
                00185 ;-------------------------------------------------------------------------
001E            00186 DisplayOff
001E 05E6       00187       bsf     PORTB,LEFT_OFF        ; turn off the left digit
001F 05C6       00188       bsf     PORTB,RIGHT_OFF       ; turn off the right digit
0020 0800       00189       retlw   0
                00190 ;
                00191 ;*************************************************************************
                00192 ;  include files
0200            00193       org             200
0200            00194 BeginMeasure
                00195       include         "measure.asm"
                00001             TITLE         "IR Receiver output measurement routine V0.07"
                00002 ;
                00003 ;*************************************************************************
                00004 ; File Name :       MEASURE.ASM
                00005 ;*************************************************************************
                00006 ;       Author:     William G. Grimm
                00007 ;       Company:    Microchip Technology
                00008 ;       Revision:   V0.07
                00009 ;       Date:       March 31, 1996
                00010 ;       Assembler:  MPASM version 1.21
                00011 ;
                00012 ;*************************************************************************
                00013 ;   Revision History:
                00014 ;
                00015 ;
                00016 ;   V0.01     Original  January 9, 1995
                00017 ;
                00018 ;   V0.02     Added overflow indication January 12, 1996
                00019 ;
                00020 ;   V0.03     Modified to conform to Microchip specifications
                00021 ;             February 17, 1996
                00022 ;
                00023 ;   V0.04     Modified for new hardware March 19, 1996
                00024 ;
                00025 ;   V0.05     Changable option register and delay added
                00026 ;             March 20, 1996
                00027 ;
                00028 ;   V0.06     Added code that creates 1/8 second time out
                00029 ;             for all options 1 to 7, 0 gets a 1/16 second time out.
                00030 ;             March 26, 1996
                00031 ;
                00032 ;   V0.07     Improved roll over detection for long gaps and pulses
                00033 ;             March 27, 1996
                00034 ;
                00035 ;   V0.08     Changed to header file March 31, 1996
                00036 ;
                00037 ;   V0.09     Fixed bug that kept jumper 1 from being read
                00038 ;
                00039 ;
                00040 ;*************************************************************************
                00041 ; Assembly options
                00042           LIST      P=16C57,r=hex,N=75,C=120,T=ON
                00043 ;
                00044 ;*************************************************************************
                00045 ;
                00046 ;
                00047 ;
                00048 ;*************************************************************************
                00049 ;*************************************************************************
                00050 ;
                00051 ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                00052          SUBTITL "RAM and Flag definitions."
                00053 ;*************************************************************************
                00054 ; file memory location definitions
```

```
                00055 ;***********************************************************************
                00056 ;
                00057 ;  full byte file memory locations
                00058 ;
00000008        00059 START_COUNT   EQU   08        ; TMR0 value at previous IR rcvr transition
00000009        00060 TR_COUNT      EQU   09        ; transition being read
                00061
0000000B        00062 TIMERM        EQU   0b        ; Bit5 = 1/4 sec, Bit1 = 16 millisecs.
0000000C        00063 FLAG          EQU   0c        ; program flags
0000000D        00064 SCALE_RECORD  EQU   0d        ; prescaler value is stored here
                00065 ;     LEFT_DIGIT    EQU   0E  ; defined in main routine
                00066 ;     RIGHT_DIGIT   EQU   0F
                00067                               ; Files 10h-1fh,30h-3fh,50h-5fh,70h-7fh
                00068                               ; are used to store IR pulse and gap lengths.
                00069 ;
                00070 ;
                00071 ;
                00072 ;
                00073 ;DEFINE FLAG REG FUNCTION:
                00074 ;       BIT # 7|6|5|4|3|2|1|0|
                00075 ;--------------|-|-|-|-|-|-|-|
                00076 ;              | | | | | | |Y| -->
                00077 ;              | | | | | |Y| | --> Eighth second flag.
                00078 ;              | | | | |Y| | | --> used for math, TMR0 overdue for reload
                00079 ;              | | | |Y| | | | --> the Value START_COUNT is new
                00080 ;              | | |Y| | | | | --> measurement has overflowed
                00081 ;              | |Y| | | | | | --> Value of last IR  bit received.
                00082 ;              |Y| | | | | | | --> if set memory is full, stop reading
                00083 ;             Y| | | | | | | | -->
                00084 ;
00000001        00085 _8TH_SEC          EQU 1
00000002        00086 OVERDUE           EQU 2
00000003        00087 NEW_START_COUNT   EQU 3
00000004        00088 OVERFLOW          EQU 4
00000005        00089 LAST_IR           EQU 5
00000006        00090 ALL_DONE          EQU 6
                00091 ;
                00092 ;
                00093 ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                00094         SUBTITL "Constant definitions."
                00095 ; ***********************************************************************
                00096 ; Definition of program constants
                00097 ;
00000020        00098 SKIP_NUM EQU d'32'                ; readings to skip before filing them
00000000        00099 OPTION_MASK   EQU    B'00000000'  ; SET UP PRESCALER, WDT on 18msec.
                00100                                  ; lowest three bits must be zero to not
                00101                                  ; overwrite the prescaler dialed in
                00102                                  ; externally
                00103 ;
                00104 ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                00105         SUBTITL "Timer routines."
                00106 ; ***********************************************************************
                00107 ; Timer servicing routine
                00108 ;    Called every 1.6 to 8 milliseconds this clears the
                00109 ;    watch dog, reloads TMR0
                00110 ;    and keeps track of relative time.
                00111 ;
0200            00112 ServiceTimerM
0200 0C83       00113       movlw   MSEC8               ;TMR0 = 8 milliseconds.
0201 01C1       00114       addwf   TMR0,W              ; Add overflow amount.
0202 0021       00115       movwf   TMR0                ;       /
0203 0004       00116       clrwdt
0204 076C       00117       btfss   FLAG,NEW_START_COUNT ; find if measured length is too long
0205 058C       00118       bsf     FLAG,OVERFLOW       ; length is too long to measure
0206 046C       00119       bcf     FLAG,NEW_START_COUNT ; set the flag indicating the reload
0207 02AB       00120       incf    TIMERM,F            ; increment the timer
```


© 1997 Microchip Technology Inc.

```
0208 090F  00121      call    TimerLookup          ; get the maximum count
0209 018B  00122      xorwf   TIMERM,W             ; see if maximum count is here
020A 0743  00123      btfss   STATUS,Z             ; Is maximum count there?
020B 0800  00124      retlw   0                    ; not there return
020C 006B  00125      clrf    TIMERM               ; reset the timer
020D 052C  00126      bsf     FLAG,_8TH_SEC        ; Set the 1/8 sec flag.
020E 0800  00127      retlw   0                    ; reset and ready for 1/8 sec chores
           00128 ;
020F       00129 TimerLookup
020F 020D  00130      movf    SCALE_RECORD,W       ; bring in the record of the option
0210 0E07  00131      andlw   7                    ; ensure lookup table is not overjumped
0211 01E2  00132      addwf   PCL,F                ; look up the proper timer overflow
0212 0800  00133      retlw   0                    ; this will only get 1/16 sec time out
0213 0800  00134      retlw   0                    ; option = 1
0214 0880  00135      retlw   b'10000000'          ; option = 2
0215 0840  00136      retlw   b'01000000'          ; option = 3
0216 0820  00137      retlw   b'00100000'          ; option = 4
0217 0810  00138      retlw   b'00010000'          ; option = 5
0218 0808  00139      retlw   b'00001000'          ; option = 6
0219 0804  00140      retlw   b'00000100'          ; option = 7
           00141 ;
           00142 ;
           00143 ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
           00144          SUBTITL "IR  counter."
           00145 ;
           00146 ;
           00147 ; ************************************************************************
           00148 ; IR  Receiver routines
           00149 ;
           00150 ;------------------------------------------------------------------------
           00151 ; ReadReceiver
           00152 ;  Second part of the IR receiver.  It takes the present count of the
           00153 ;  TMR0 and subtracts the count recorded when the receiver output
           00154 ;  went high (START_COUNT) to find the dark pulse duration.  In that duration
           00155 ;  will be encoded the 1, 0, HOLD, or attention.
           00156 ;------------------------------------------------------------------------
021A       00157 ReadReceiverHi
021A 05AC  00158      bsf     FLAG,LAST_IR         ; record the IR receiver state
021B 02A9  00159      incf    TR_COUNT,F
021C 0509  00160      bsf     TR_COUNT,0           ; Times when IR rcvr is Lo are recorded
           00161                                   ; in odd numbered locations
021D 0C3F  00162      movlw   3fh                  ; bring in highest valid address in
021E 0189  00163      xorwf   TR_COUNT,W           ; TR_count see if highest count is in
021F 0643  00164      btfsc   STATUS,Z             ; skip if not highest address
0220 05CC  00165      bsf     FLAG,ALL_DONE
           00166                                   ; set all done flag to stop reading
0221 0A25  00167      goto    TimeIRReceiver
           00168 ;
0222       00169 ReadReceiverLo
0222 04AC  00170      bcf     FLAG,LAST_IR         ; record the IR value
0223 02A9  00171      incf    TR_COUNT,F
0224 0409  00172      bcf     TR_COUNT,0           ; Times when IR rcvr is Hi are recorded
           00173                                   ; in even numbered locations
           00174 ;
           00175                                   ; Calc the length of the dark pulse,
           00176                                   ; length of time receiver was high.
           00177                                   ; (placed in START_COUNT)
0225       00178 TimeIRReceiver
0225 0208  00179      movf    START_COUNT,W        ; bring in the start measurement
0226 0081  00180      subwf   TMR0,W               ; subtract the final from the start
0227 0028  00181      movwf   START_COUNT          ; gap or pulse length is now in
           00182                                   ; START_COUNT, must be checked
0228 0C83  00183      movlw   MSEC8                ; Base number of TMR0 count.
0229 0088  00184      subwf   START_COUNT,W        ; Subtract the base count of TMR0
022A 0603  00185      btfsc   STATUS,C             ; skip the store and toss value if neg.
022B 0028  00186      movwf   START_COUNT          ; value was positive, store
```

```
                00187 ;
022C            00188 MathDone
022C 06E9       00189       btfsc     TR_COUNT,7          ; check to see if in delay
022D 0A3A       00190       goto      DoNotStore          ; do not store the value if in delay
                00191 ;
                00192                                     ; format for FSR
022E 0209       00193       movf      TR_COUNT,W          ; Setup place the count will be stored
022F 0E0F       00194       andlw     0f                  ; reduce to file location
0230 0024       00195       movwf     FSR                 ; place file address in FSR
0231 0584       00196       bsf       FSR,4               ; set to place in upper file group
0232 0689       00197       btfsc     TR_COUNT,4          ; set bank bit 0
0233 05A4       00198       bsf       FSR,5               ; /
0234 06A9       00199       btfsc     TR_COUNT,5          ; set bank bit 1
0235 05C4       00200       bsf       FSR,6               ; /
                00201 ;
0236 0CFF       00202       movlw     0ffh                ; bring in the overflow indication
0237 078C       00203       btfss     FLAG,OVERFLOW       ; skip loading of result if overflowed
0238 0208       00204       movf      START_COUNT,W       ; bring in the measurement
0239 0020       00205       movwf     INDF                ; store it for display using
                00206                                     ; indirect addressing
023A            00207 DoNotStore
023A 0201       00208       movf      TMR0,W              ; bring in the count now
023B 0028       00209       movwf     START_COUNT         ; store it for next time
023C 056C       00210       bsf       FLAG,NEW_START_COUNT
                00211                                     ; set ind flag that START_COUNT is new.
                00212                                     ; this flag is used to determine if the
                00213                                     ; pulse has gone on too long to measure
023D 048C       00214       bcf       FLAG,OVERFLOW       ; clear any overflow indication
                00215 ;
023E 0800       00216       retlw     0
                00217 ;
                00218 ;
                00219 ;*****************************************************************************
                00220 ; The following code segments are called by the executive
                00221 ; every 1/8 second and every two seconds
                00222 ;
023F            00223 EighthSecondChores                  ; all that needs doing every 1/8 sec
                00224                                     ; can be placed in this subroutine
023F 042C       00225       bcf       FLAG,_8TH_SEC       ; clear the time out flag
                00226 ;
0240 0246       00227       comf      PORTB,W             ; read the dial settings
                00228                                     ; the requested memory location is in W
0241 0024       00229       movwf     FSR
                00230                                     ; Following formats the FSR to point to
                00231                                     ; the selected file w/ the IR pulse
0242 04C4       00232       bcf       FSR,6               ; or gap length
0243 06A4       00233       btfsc     FSR,5               ; move bit 5 to 6
0244 05C4       00234       bsf       FSR,6               ; if 5 was 1, set 6
0245 04A4       00235       bcf       FSR,5
0246 0684       00236       btfsc     FSR,4               ; move bit 4 to 5
0247 05A4       00237       bsf       FSR,5               ; if 4 was high, set bit 5 of fsr
0248 0584       00238       bsf       FSR,4               ; Format for FSR, upper bank of bytes
                00239 ;
0249 04A3       00240       bcf       STATUS,PA0          ; get ready to call from page 1
024A 04C3       00241       bcf       STATUS,PA1          ;  /
                00242 ;
024B 0380       00243       swapf     INDF,W              ; bring in IR measurement to be disp'd
024C 0900       00244       call      LookUpDigit
024D 002E       00245       movwf     LEFT_DIGIT          ; display more significant digit
                00246 ;
024E 0200       00247       movf      INDF,W              ; bring in IR measurement to be disp'd
024F 0900       00248       call      LookUpDigit
0250 002F       00249       movwf     RIGHT_DIGIT         ; display less significant digit
                00250 ;
                00251  if BeginMeasure==200              ; return the bits to this page
0251 05A3       00252       bsf       STATUS,PA0          ; page 1
```


© 1997 Microchip Technology Inc.

```
0252 04C3  00253      bcf      STATUS,PA1          ; /
           00254  endif
           00255  if BeginMeasure==400
           00256      bcf      STATUS,PA0          ; page 2
           00257      bsf      STATUS,PA1          ; /
           00258  endif
           00259  if BeginMeasure==600
           00260      bsf      STATUS,PA0          ; page 3
           00261      bsf      STATUS,PA1          ; /
           00262  endif
           00263 ;
0253 0245  00264      comf     PORTA,W             ; bring in the req'd prescale value
           00265                                  ; from the dial, reverse sense
0254 0E07  00266      andlw    7                   ; AND w/ highest possible prescale value
0255 018D  00267      xorwf    SCALE_RECORD,W      ; compare the prescale dial setting
           00268                                  ; with the original one
0256 0743  00269      btfss    STATUS,Z            ; skip if the same
0257 0A61  00270      goto     StartMeasure        ; restart the application if different
0258 0AAD  00271      goto     DoneEighthSecondChores
           00272 ;
           00273 ;
0259       00274 ClearRam                         ; clears memory at reset
0259 0024  00275      movwf    FSR                 ; place in fsr for indirect addressing.
025A 003F  00276      movwf    1fh                 ; when zero, memory init is done.
025B       00277 MemoryInitLoop
025B 02A4  00278      incf     FSR,F               ; increment to the next memory location
           00279                                  ; to be initialized.
025C 0060  00280      clrf     INDF                ; clear memory location.
025D 021F  00281      movf      1fh,w              ; Has top memory location zeroed yet?
025E 0743  00282      btfss    STATUS,Z            ;    /
025F 0A5B  00283      goto     MemoryInitLoop      ; /
0260 0800  00284      retlw    0
           00285 ;
           00286 ;
           00287 ;*****************************************************************************
           00288 ;          Start HERE.
           00289 ;*****************************************************************************
0261       00290 StartMeasure
0261 006C  00291      clrf     FLAG                ; Clear out flag bank 1.
           00292 ;
0262 006B  00293      clrf     TIMERM              ; restart the TIMERM at 0
           00294 ;
0263 0069  00295      clrf     TR_COUNT            ; initialize memory counter
           00296 ;
0264 0C0F  00297      movlw    0f                  ; start zeroing at memory location 10h
0265 0959  00298      call     ClearRam            ; clear the first bank of memory
           00299 ;
0266 0C2F  00300      movlw    2f                  ; start zeroing at memory location 10h.
0267 0959  00301      call     ClearRam            ; clear the second bank of memory
           00302 ;
0268 0C4F  00303      movlw    4f                  ; start zeroing at memory location 10h
0269 0959  00304      call     ClearRam            ; clear the third bank of memory
           00305 ;
026A 0C6F  00306      movlw    6f                  ; start zeroing at memory location 10h
026B 0959  00307      call     ClearRam            ; clear the fourth bank of memory
           00308 ;
026C 0C0F  00309      movlw    A_CONFIG            ; setup for PORTA, in loop so
           00310                                  ; micocontroller will never forget
026D 0005  00311      tris     PORTA               ; inputs on bit 0.
026E 0C3F  00312      movlw    B_CONFIG
026F 0006  00313      tris     PORTB               ; PORTB has outputs on bits 0,6,7;
           00314                                  ; inputs on bits 1, 2, 3, 4, and 5.
           00315 ;
0270 05E6  00316      bsf      PORTB,LEFT_OFF      ; turn off both digits to read jumpers
0271 05C6  00317      bsf      PORTB,RIGHT_OFF     ; /
0272 0C1E  00318      movlw    C_CONFIG1           ; configuration to read from PORTC
```

```
0273 0007  00319      tris     PORTC              ; configure PORTC to read the bits
0274 0CE0  00320      movlw    -(SKIP_NUM)        ; let inputs settle, bring in skip numb
0275 0727  00321      btfss    PORTC,SW1          ; skip if jumper 1 is not installed
0276 0029  00322      movwf    TR_COUNT           ; move the skip number to file pointer
           00323 ;
0277 0C00  00324      movlw    C_CONFIG2          ; bring in config to use PORTC for disp
0278 0007  00325      tris     PORTC              ; PORTC is normally all outputs
           00326 ;
0279 0245  00327      comf     PORTA,W            ; bring in the requested prescale
           00328                                 ; value from the dial, reverse sense
027A 0E07  00329      andlw    7                  ; AND w/ highest possible prescale value
027B 002D  00330      movwf    SCALE_RECORD       ; record the value of the prescaler
027C 0D00  00331      iorlw    OPTION_MASK        ; Setup prescaler for TMR0, WDT on 18ms.
027D 0002  00332      option                      ; /
           00333 ;
           00334 ;
027E 0CBF  00335      movlw    HIPHEN             ; Disp that unit waiting for dark cond's
027F 0027  00336      movwf    PORTC              ; put the Hiphen on right digit
0280 05E6  00337      bsf      PORTB,LEFT_OFF     ; turn off left digit
0281 04C6  00338      bcf      PORTB,RIGHT_OFF    ; turn ON right digit
           00339 ;
0282 0C83  00340      movlw    MSEC8              ;TMR0 = 8 mSEC
0283 0021  00341      movwf    TMR0               ;       /
           00342 ;
0284       00343 SettlingLoop
0284 0C83  00344      movlw    MSEC8              ; Check for overflow.
0285 0081  00345      subwf    TMR0,W             ; SEE IF TMR0 < MSEC8,
0286 0603  00346      btfsc    STATUS,C           ; If TMR0 < MSEC8, Overflow.
0287 0A84  00347      goto     SettlingLoop       ; No overflow, no carry, loop.
0288 0900  00348      call     ServiceTimerM      ; Keep time and reload time keeper.
0289 0765  00349      btfss    PORTA,IR           ; IR receiver quiet?
028A 006B  00350      clrf     TIMERM             ; not quiet, reset timer
028B 072C  00351      btfss    FLAG,_8TH_SEC      ; Allow out of loop if quiet for 1/8sec
028C 0A84  00352      goto     SettlingLoop       ; not quiet long enough yet
028D 006C  00353      clrf     FLAG               ; re-clear all of the flags
028E 05AC  00354      bsf      FLAG,LAST_IR       ; set the flag, receiver is now hi
           00355 ;
           00356 ;
028F 0CBF  00357      movlw    HIPHEN             ; Display that unit is ready to receive
0290 0027  00358      movwf    PORTC              ; put the Hiphen on right digit
0291 04E6  00359      bcf      PORTB,LEFT_OFF     ; turn on left digit
0292 05C6  00360      bsf      PORTB,RIGHT_OFF    ; turn OFF right digit
           00361 ;
0293 093A  00362      call     DoNotStore         ; setup timer last read for first read
           00363 ;
           00364 ;************ Main loop Starts here. *************
0294       00365 Main
           00366 ;
           00367 ;
0294       00368 InnerLoop
0294 07CC  00369      btfss    FLAG,ALL_DONE      ; update display only if memory is full
0295 0A9E  00370      goto     CheckIr            ; not full, keep reading the IR rcvr
0296 04A3  00371      bcf      STATUS,PA0         ; get ready to call from page 1
0297 04C3  00372      bcf      STATUS,PA1         ;  /
0298 0912  00373      call     UpdateDisplay      ; rotate power to the next display digit
           00374 ;
           00375  if BeginMeasure==200           ; return the bits to this page
0299 05A3  00376      bsf      STATUS,PA0         ; page 1
029A 04C3  00377      bcf      STATUS,PA1         ; /
           00378  endif
           00379  if BeginMeasure==400
           00380      bcf      STATUS,PA0         ; page 2
           00381      bsf      STATUS,PA1         ; /
           00382  endif
           00383  if BeginMeasure==600
           00384      bsf      STATUS,PA0         ; page 3
```


© 1997 Microchip Technology Inc.

```
                00385       bsf     STATUS,PA1          ; /
                00386   endif
029B 07C6       00387       btfss   PORTB,RIGHT_OFF     ; skip if the right digit is off
029C 04E7       00388       bcf     PORTC,DP            ; lite the decimal to show read taken
029D 0AA6       00389       goto    ReadDone            ; memory is full, done reading receiver
                00390 ;
029E            00391 CheckIr
029E 0665       00392       btfsc   PORTA,IR            ; ?IR receiver not recv'g an IR burst?
029F 06AC       00393       btfsc   FLAG,LAST_IR        ; was it receiving a burst last time?
02A0 02A2       00394       incf    PCL,F               ; Not either, skip next instruction
02A1 091A       00395       call    ReadReceiverHi      ; Record TMR0 value when the lo to hi
                00396                                   ; transition came from the receiver
                00397 ;
02A2 0765       00398       btfss   PORTA,IR            ; ?IR receiver receiving an IR burst?
02A3 07AC       00399       btfss   FLAG,LAST_IR        ; was it not receiving burst last time?
02A4 02A2       00400       incf    PCL,F               ; Not either skip next instruction
02A5 0922       00401       call    ReadReceiverLo      ; read the new information
                00402 ;
02A6            00403 ReadDone
                00404 ;
02A6 0C83       00405       movlw   MSEC8               ; Check for overflow.
02A7 0081       00406       subwf   TMR0,W              ; SEE IF TMR0 < MSEC8,
02A8 0603       00407       btfsc   STATUS,C            ; If TMR0 < MSEC8, Overflow.
02A9 0A94       00408       goto    InnerLoop           ; No overflow, no carry, loop.
02AA 0900       00409       call    ServiceTimerM       ; Keep time and reload time keeper.
                00410 ;
02AB 062C       00411       btfsc   FLAG,_8TH_SEC       ; check for 1/8 second time out
02AC 0A3F       00412       goto    EighthSecondChores
                00413                                   ; anything that needs doing every 1/8sec
                00414                                   ; can go in this subroutine
02AD            00415 DoneEighthSecondChores
                00416 ;
02AD 0A94       00417       goto    Main
                00418 ;
                00419 ;
                00420
0400            00196       org             400
0400            00197 BeginIr6121
                00198       include         "ir6121.asm"
                00001               TITLE       "IR-NEC6121 format Remote Control Detector V0.02"
                00002               SUBTITL     "Comments documentation and history"
                00003 ;
                00004 ;*****************************************************************************
                00005 ; File Name :       IR6121.ASM
                00006 ;*****************************************************************************
                00007 ;       Author:     William G. Grimm
                00008 ;       Company:    Microchip Technology
                00009 ;       Revision:   V0.02
                00010 ;       Date:       February 27, 1996
                00011 ;       Assembler:  MPASM version 1.21
                00012 ;
                00013 ;*****************************************************************************
                00014 ;   Revision History:
                00015 ;
                00016 ;
                00017 ;   V0.01    Original  February 27, 1996
                00018 ;
                00019 ;   V0.02    Converted to Ap-note format and made into a header
                00020 ;            file March 28, 1996
                00021 ;*****************************************************************************
00000005        00022 OPTION_CODE   EQU    B'00000101'     ;SET UP PRESCALER, WDT on 18msec.
                00023 ;*****************************************************************************
                00024 ;
                00025 ;*****************************************************************************
                00026 ; file memory location definitions
                00027 ;*****************************************************************************
```

```
                        00028 ;
                        00029 ;  full byte file memory locations
                        00030 ;
0000000B                00031 TIMER         EQU   0b        ; Bit 5 = 1/4 second, Bit 1 = 16 millisecs.
0000000C                00032 TEMP          EQU   0c        ; temporary file storage
                        00033 ;
                        00034 ;    LEFT_DIGIT    EQU   0e   ; defined in irmain
                        00035 ;    RIGHT_DIGIT   EQU   0f   ; defined in irmain
00000011                00036 READ_LH       EQU   11        ; Low to high reading is stored here.
00000012                00037 IR_STATE      EQU   12        ; Which bit is coming in.
00000013                00038 IR_BYTE13     EQU   13        ; First byte for collecting inputs.
00000014                00039 IR_BYTE24     EQU   14        ; Second byte for collecting inputs.
                        00040 ;
00000018                00041 FLAG2         EQU   18        ; flag bank 2
00000019                00042 FLAG3         EQU   19        ; flag bank 3
0000001A                00043 C_BYTE_1      EQU   1A        ; Memory location def's for storing inputs
0000001B                00044 C_BYTE_2      EQU   1B
0000001C                00045 BYTE_1        EQU   1C
0000001D                00046 BYTE_2        EQU   1D
0000001E                00047 BYTE_3        EQU   1E
0000001F                00048 BYTE_4        EQU   1F
                        00049 ;
                        00050 ;DEFINE FLAG2 REG FUNCTION:
                        00051 ;       BIT # 7|6|5|4|3|2|1|0|
                        00052 ;--------------|-|-|-|-|-|-|-|
                        00053 ;              | | | | | | |Y| --> Command Ready.
                        00054 ;              | | | | | |Y| | --> Command in process.
                        00055 ;              | | | | |Y| | | --> Most Significant bit of time stamp.
                        00056 ;              | | | |Y| | | | --> HOLD is active
                        00057 ;              | | |Y| | | | | --> Four bytes have been recv'd ok
                        00058 ;              | |Y| | | | | | --> Value of last IR bit received.
                        00059 ;              |Y| | | | | | | --> A Valid hold received < 1/4 sec ago.
                        00060 ;             Y| | | | | | | | -->
                        00061 ; Y = DEFINED AS SHOWN (0/1)
                        00062 ;
00000006                00063 HOLD_RCVD           EQU 6
00000005                00064 LAST_IR_STATE       EQU 5
00000004                00065 KEY_READY           EQU 4
00000003                00066 HOLD                EQU 3
00000002                00067 STAMP_MSB           EQU 2
00000001                00068 CMD_PEND            EQU 1  ; A channel command is pending.
00000000                00069 CMD_RDY             EQU 0  ; A channel command is ready.
                        00070 ;
                        00071 ;DEFINE FLAG3 REG FUNCTION:
                        00072 ;       BIT # 7|6|5|4|3|2|1|0|
                        00073 ;--------------|-|-|-|-|-|-|-|
                        00074 ;              | | | | | | |Y| --> Quarter second flag.
                        00075 ;              | | | | | |Y| | --> Eighth second flag.
                        00076 ;              | | | | |Y| | | --> Two second flag.
                        00077 ;              | | | |Y| | | | -->
                        00078 ;              | | |Y| | | | | -->
                        00079 ;              | |Y| | | | | | -->
                        00080 ;              |Y| | | | | | | -->
                        00081 ;             Y| | | | | | | | -->
                        00082 ; Y = DEFINED AS SHOWN (0/1)
                        00083 ;
                        00084 ;
00000000                00085 _4TH_SEC      EQU 0
00000001                00086 EIGTH_SEC     EQU 1
00000002                00087 TWO_SEC       EQU 2
                        00088 ;
                        00089 ;^^^
```

```
                00094 ;
0000001A        00095 HOLD_MIN      EQU  1a         ; Changed from 1e to minimize intermittency.
00000027        00096 HOLD_MAX      EQU  27         ; Changed from 23 to minimize intermittency.
00000038        00097 HEAD_MIN      EQU  38
00000048        00098 HEAD_MAX      EQU  48
0000001F        00099 ONE_MAX       EQU  1f
0000000E        00100 ZERO_MAX      EQU  0e
                00101 ;
                00102 ;
                00103 ;
                00104 ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                00105         SUBTITL "Timer Routines."
                00106 ;*************************************************************************
                00107 ; Timer servicing routine
                00108 ;    Called every 8 milliseconds this clears the
                00109 ;    watch dog, reloads the real time clock counter
                00110 ;    and keeps track of relative time.
                00111 ;
0400            00112 SvcTimer
0400 0C83       00113       movlw    MSEC8              ;TMR0 = 8 milliseconds.
0401 01C1       00114       addwf    TMR0,W             ; Add overflow amount.
0402 0021       00115       movwf    TMR0               ;       /
0403 0004       00116       clrwdt
0404 03EB       00117       incfsz   TIMER,F            ; Increment the timer, Skip to two sec
                00118                                   ; set up if it rolls over.
0405 0A08       00119       goto     CheckMatch         ; Go to other possible set ups.
0406 0559       00120       bsf      FLAG3,TWO_SEC      ; Set the 2 second flag.
                00121 ;
0407 0800       00122       retlw    0                  ; sync serviced.
0408            00123 CheckMatch                        ; 1/8 and 1/4 sec flags are staggered
                00124                                   ; for more eff use of processor time.
0408 020B       00125       movf     TIMER,W            ; Bring in the timer.
0409 0E0F       00126       andlw    b'00001111'        ; Check lower bits.
040A 0F02       00127       xorlw    d'2'               ; 1/8sec chores called when lo nibble=2.
040B 0643       00128       btfsc    STATUS,Z           ; Was the low nibble not 2?
040C 0539       00129       bsf      FLAG3,EIGTH_SEC    ; No! it was 2, Set the 1/8 sec flag.
040D 020B       00130       movf     TIMER,W            ; Bring in the timer.
040E 0E1F       00131       andlw    b'00011111'        ; Check five lower bits.
040F 0F19       00132       xorlw    0x19               ; 1/4 second chores called every 0x19.
0410 0643       00133       btfsc    STATUS,Z           ; Was the low five bits not 0x19?
0411 0519       00134       bsf      FLAG3,_4TH_SEC     ; No! it was 0x19, Set the 1/4sec flag.
0412 0800       00135       retlw    0                  ; matches checked, return.
                00136 ;
                00137 ;
                00138 ;
                00139 ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                00140         SUBTITL "6121 type IR remote control reader."
                00141 ; *************************************************************************
                00142 ; The following reads the IR transmitter.
                00143 ; When the IR transmitter is being read,
                00144 ; This routine takes control of the clocks
                00145 ; and suspends all other functions.
                00146 ;
                00147 ; *************************************************************************
                00148 ; IR Receiver routine
                00149 ;
                00150 ;
                00151 ;-------------------------------------------------------------------------
                00152 ; ReadAddr
                00153 ;   This routine places the first two bytes received in temporary
                00154 ;   locations.  Normally this routine would be configured to detect
                00155 ;   weather or not the received command was ment for this equipment
                00156 ;-------------------------------------------------------------------------
0413            00157 ReadAddr
0413 0213       00158       movf     IR_BYTE13,W        ; bring in the first complete byte read
0414 003A       00159       movwf    C_BYTE_1           ; store it in the contingent first byte
```

```
0415 0214  00160      movf     IR_BYTE24,W         ; bring in the second complete byte read
0416 003B  00161      movwf    C_BYTE_2            ; store it in the contingent second byte
0417 0800  00162      retlw    0                   ; command.
           00163 ;
           00164 ;-------------------------------------------------------------------------
           00165 ; ReadCommand
           00166 ;   This routine places the third and fourth bytes in memory locations
           00167 ;   so they can be displayed.  The first two bytes are transferred from
           00168 ;   their trmporary locations to locations where they too can be
           00169 ;   displayed.  Normally this routine would be configured to decode
           00170 ;   the appropriate action from the received number.  the third and fourth
           00171 ;   bytes are always complements of each other in this format.  Typically
           00172 ;   a complementary check of these two bytes is done at this point in the IR
           00173 ;   reception
           00174 ;-------------------------------------------------------------------------
0418       00175 ReadCommand
0418 021A  00176      movf     C_BYTE_1,W      ; bring in the first complete byte read
0419 003C  00177      movwf    BYTE_1          ; store it to be disp'd as actual first byte
041A 021B  00178      movf     C_BYTE_2,W      ; bring in the first complete byte read
041B 003D  00179      movwf    BYTE_2          ; store it to be disp'd as actual 2nd byte
041C 0213  00180      movf     IR_BYTE13,W     ; bring in the third complete byte read
041D 003E  00181      movwf    BYTE_3          ; store it to be displayed as the 3rd byte
041E 0214  00182      movf     IR_BYTE24,W     ; bring in the fourth complete byte read
041F 003F  00183      movwf    BYTE_4          ; store it to be displayed as the 4th byte
0420 0598  00184      bsf      FLAG2,KEY_READY ; Good set received
0421 0A48  00185      goto     LogHold         ; Activate the hold for the first pass.
           00186 ;
           00187 ;
           00188 ;-------------------------------------------------------------------------
           00189 ; ReadReceiver
           00190 ;  Second part of the IR receier.  It takes the present count of the
           00191 ;  RTCC and subtracts the count recorded when the receiver output
           00192 ;  went high (READ_LH) to find the dark pulse duration.  In that duration
           00193 ;  will be encoded the 1, 0, HOLD, or attention.
           00194 ;-------------------------------------------------------------------------
0422       00195 ReadReceiver
0422 04B8  00196      bcf      FLAG2,LAST_IR_STATE ; Record that the IR receiver output
           00197                                   ; is now high
           00198                                   ; Calc the length of the dark pulse,
           00199                                   ; length of time receiver was high.
           00200                                   ; (placed in READ_LH)
0423 0211  00201      movf     READ_LH,W           ; bring in the start measurement
0424 0081  00202      subwf    TMR0,W              ; subtract the final from the start
0425 0031  00203      movwf    READ_LH             ; gap or pulse length is now in
           00204                                   ; READ_LH, must be checked
0426 0C83  00205      movlw    MSEC8               ; Base number of TMR0 count.
0427 0091  00206      subwf    READ_LH,W           ; Subtract the base count of TMR0
0428 0603  00207      btfsc    STATUS,C            ; skip the store and toss value if neg
0429 0031  00208      movwf    READ_LH             ; value was positive, store
           00209 ;
042A       00210 Ir6121MathDone
           00211 ;
042A 0678  00212      btfsc    FLAG2,HOLD          ; is it now looking for holds?
042B 0A40  00213      goto     LookForHold         ; look for HOLD
           00214 ;
042C 094B  00215      call     LookForAttentionGap ; look for an attention dark pulse
042D 01E2  00216      addwf    PCL,F               ; skip if a 1 was ret'd, no atten pulse
042E 0800  00217      retlw    0                   ; a 0 ret'd, ATTEN pulse found, return
           00218 ;
042F 0C1F  00219      movlw    ONE_MAX          ; Test for the max length of one.
0430 0091  00220      subwf    READ_LH,W        ; If no carry gen'd, A valid 1 is found
0431 0603  00221      btfsc    STATUS,C         ; No carry means the reading is below max
0432 0A5E  00222      goto     ResetIR          ; IR no good, Above maximum is invalid.
           00223 ;
0433 0C0E  00224      movlw    ZERO_MAX         ; Test for the max length of Zero.
0434 0091  00225      subwf    READ_LH,W        ; If no carry gen'd, A valid 0 is found.
```


© 1997 Microchip Technology Inc.

```
                00226                                   ; the carry now has the newly received bit
                00227                                   ; shift the bit into the proper location
                00228 ;
0435 0772       00229       btfss   IR_STATE,3          ; Every 8 states result in dest changes
0436 0333       00230       rrf     IR_BYTE13,F         ; this bit is a part of IR byte 1 or 3
0437 0672       00231       btfsc   IR_STATE,3          ;   /
0438 0334       00232       rrf     IR_BYTE24,F         ; this bit is a part of ir byte 2 or 4
                00233 ;
0439 0C01       00234       movlw   1                   ; Get ready to add one to the IR STATE
043A 01F2       00235       addwf   IR_STATE,F          ; inc the state setting half carry bits
043B 0723       00236       btfss   STATUS,DC           ; skip if digit carry generated
043C 0800       00237       retlw   0                   ; all done reading for now
043D 07B2       00238       btfss   IR_STATE,5          ; check to determine if the 1st and 2nd
                00239                                   ; bytes or 3rd and 4th bytes are now ready
043E 0A13       00240       goto    ReadAddr            ; First and second byte ready.
043F 0A18       00241       goto    ReadCommand         ; Third and fourth byte ready.
                00242 ;
                00243 ;-------------------------------------------------------------------------
                00244 ; LookForHold
                00245 ;  Reads the length of the received dark pulse and determines if
                00246 ;  a valid HOLD pulse has been received
                00247 ;-------------------------------------------------------------------------
0440            00248 LookForHold
0440 0C1A       00249       movlw   HOLD_MIN            ; Find if between hold and one.
0441 0091       00250       subwf   READ_LH,W           ; IF no carry is gen'd, The read is between
0442 0703       00251       btfss   STATUS,C            ; HOLD and one and as such, invalid.
0443 0800       00252       retlw   0                   ; Return to main routine from invalid read
0444 0C27       00253       movlw   HOLD_MAX            ; Test for the max length of HOLD.
0445 0091       00254       subwf   READ_LH,W           ; If no carry is gen'd, get a valid hold
0446 0603       00255       btfsc   STATUS,C            ;
0447 0800       00256       retlw   0
0448            00257 LogHold
0448 0578       00258       bsf     FLAG2,HOLD          ; valid HOLD received
0449 05D8       00259       bsf     FLAG2,HOLD_RCVD     ; clear bit for the next hold condition
044A 0800       00260       retlw   0
                00261 ;
                00262 ;-------------------------------------------------------------------------
                00263 ; LookForAttentionGap
                00264 ;  Reads the length of the received dark pulse and determines if
                00265 ;  a valid attention pulse has been received
                00266 ;-------------------------------------------------------------------------
044B            00267 LookForAttentionGap           ; look for attention dark pulse
044B 0C38       00268       movlw   HEAD_MIN        ; Find if between head and one.
044C 0091       00269       subwf   READ_LH,W       ; IF no carry is gen'd, reading is between
044D 0703       00270       btfss   STATUS,C        ; HOLD and HEAD and as such, invalid.
044E 0A55       00271       goto    CheckIRState    ; continue, no attention gap.
044F 0C48       00272       movlw   HEAD_MAX        ; Test for the max length of HEAD.
0450 0091       00273       subwf   READ_LH,W       ; If no carry is gen'd, get a valid head.
0451 0603       00274       btfsc   STATUS,C        ; A carry = a too long gap and is invalid.
0452 0A55       00275       goto    CheckIRState    ; continue, no attention gap
0453 0072       00276       clrf    IR_STATE        ; Valid Attention dark pulse.  This command
                00277                               ; starts the state machine looking for bits
0454 0800       00278       retlw   0               ; return to main routine, ATTEN found
0455            00279 CheckIRState
0455 0CE0       00280       movlw   0e0             ; load A mask to mask all counting states
0456 0152       00281       andwf   IR_STATE,W      ; compare with present state
0457 0743       00282       btfss   STATUS,Z
0458 0800       00283       retlw   0               ; not a count state, return to main routine
0459 0801       00284       retlw   01              ; counting state, look for 1's and 0's
                00285 ;
                00286 ;-------------------------------------------------------------------------
                00287 ; RecordRTCCatLowToHiTransition
                00288 ;  First part of the IR receier.  It records the time when the
                00289 ;  output of the IR receiver went from low to high.  this creates the
                00290 ;  starting time for timing an IR pulse.
                00291 ;-------------------------------------------------------------------------
```


© 1997 Microchip Technology Inc.

```
045A        00292 RecordRTCC_atLowToHiTransition
045A 05B8   00293       bsf     FLAG2,LAST_IR_STATE ; record that IR was last in dark pulse
045B 0201   00294       movf    TMR0,W              ; bring in the clock time
045C 0031   00295       movwf   READ_LH             ; record for when it goes back low
045D 0800   00296       retlw   0
            00297 ;-------------------------------------------------------------------------
            00298 ; ResetIR
            00299 ;  Resets the IR state machine to ready it for receiving IR messages.
            00300 ;-------------------------------------------------------------------------
045E        00301 ResetIR
045E 0478   00302       bcf     FLAG2,HOLD          ; not seen clear the hold
045F 04D8   00303       bcf     FLAG2,HOLD_RCVD     ; clear the bit for next hold condition
0460 0072   00304       clrf    IR_STATE            ; preset IR_STATE to -1
0461 0272   00305       comf    IR_STATE,F          ;  /
0462 0800   00306       retlw   0
            00307 ;
            00308 ;*************************************************************************
            00309 ; The following subroutines are called by the executive
            00310 ; every 1/8 second, every 1/4 second, and every two seconds
            00311 ;
0463        00312 EigthSecChores                    ; all that needs doing every 1/8 sec
            00313                                   ; can be placed in this subroutine
0463 0439   00314       bcf     FLAG3,EIGTH_SEC     ; clear the time out flag
            00315 ;
0464 0245   00316       comf    PORTA,W             ; bring in the requested prescale value
            00317                                   ; from the dial, reverse sense
0465 0E07   00318       andlw   7                   ; AND w/ highest possible prescale value
0466 0643   00319       btfsc   STATUS,Z            ; if zero, display hiphens
0467 0A7A   00320       goto    DisplayHiphens      ; was zero, display hiphens
0468 002C   00321       movwf   TEMP                ; place in temporary storage
0469 00EC   00322       decf    TEMP,F              ; dec, dial settings 1 to 4 are valid
046A 064C   00323       btfsc   TEMP,2              ; if bit2 is set dial is 5 or higher
046B 0A7A   00324       goto    DisplayHiphens      ; dial is above 5, display hiphens
046C 0C1C   00325       movlw   BYTE_1
046D 01CC   00326       addwf   TEMP,W              ; add in dial setting (between 0 and 3)
046E 0024   00327       movwf   FSR                 ; place in pointer register.
            00328                                   ; NOTE! FSR bits5,6 = clear, File page1
            00329 ;
046F 04A3   00330       bcf     STATUS,PA0          ; get ready to call from page 1
0470 04C3   00331       bcf     STATUS,PA1          ;  /
            00332 ;
0471 0380   00333       swapf   INDF,W              ; bring in IR measurement to be disp'd
0472 0900   00334       call    LookUpDigit
0473 002E   00335       movwf   LEFT_DIGIT          ; display more significant digit
            00336 ;
0474 0200   00337       movf    INDF,W              ; bring in IR measurement to be disp'd
0475 0900   00338       call    LookUpDigit
0476 002F   00339       movwf   RIGHT_DIGIT         ; display less significant digit
            00340 ;
            00341  if BeginIr6121==200              ; return the bits to this page
            00342       bsf     STATUS,PA0          ; page 1
            00343       bcf     STATUS,PA1          ; /
            00344  endif
            00345  if BeginIr6121==400
0477 04A3   00346       bcf     STATUS,PA0          ; page 2
0478 05C3   00347       bsf     STATUS,PA1          ; /
            00348  endif
            00349  if BeginIr6121==600
            00350       bsf     STATUS,PA0          ; page 3
            00351       bsf     STATUS,PA1          ; /
            00352  endif
            00353 ;
0479 0800   00354       retlw   0
047A        00355 DisplayHiphens
047A 0CBF   00356       movlw   HIPHEN              ; dial not in range, display hiphens
047B 002E   00357       movwf   LEFT_DIGIT          ; / Hiphen in left digit
```


© 1997 Microchip Technology Inc.

```
047C 002F  00358      movwf    RIGHT_DIGIT          ; / Hiphen in right digit
047D 0800  00359      retlw    0
           00360 ;
047E       00361 QuarterSecChores                   ; all that needs doing every 1/4 second
           00362                                    ; can be placed in this subroutine
047E 0678  00363      btfsc    FLAG2,HOLD           ; Check for HOLD condition still valid
047F 06D8  00364      btfsc    FLAG2,HOLD_RCVD      ; Check to see if a hold pulse has been
           00365                                    ; seen in the last 1/4 second
0480 02A2  00366      incf     PCL,F
0481 095E  00367      call     ResetIR              ; reset the IR state machine and get
           00368                                    ; ready for next
0482 04D8  00369      bcf      FLAG2,HOLD_RCVD      ; Clear the hold received flag, it is
           00370                                    ; to be set by IR controller
           00371 ;
0483 0419  00372      bcf      FLAG3,_4TH_SEC       ; clear the 1/4 second time out
0484 0800  00373      retlw    0
           00374 ;
0485       00375 TwoSecChores                       ; things done every two seconds
0485 0C0F  00376      movlw    A_CONFIG             ; setup for PORTA, in loop so
           00377                                    ; microcontroller will never forget
0486 0005  00378      tris     PORTA                ; inputs are on bits 0,1, and 2.
0487 0C3F  00379      movlw    B_CONFIG             ; PORTB inputs are not used,
0488 0006  00380      tris     PORTB                ; PORTB outputs control digit drives
0489 0C00  00381      movlw    C_CONFIG2
048A 0007  00382      tris     PORTC                ; PORTC is all outputs
048B 0498  00383      bcf      FLAG2,KEY_READY      ; Routine that would interpret the key
           00384                                    ; will clear the flag that says it is
           00385                                    ; ready
048C 0459  00386      bcf      FLAG3,TWO_SEC        ; clear the two second time out
048D 0800  00387      retlw    0
           00388 ;
           00389 ;
           00390 ;*****************************************************************************
           00391 ;          Start HERE.
           00392 ;*****************************************************************************
048E       00393 StartIr6121
048E 0985  00394      call     TwoSecChores         ; re-setup ports A, B, and C
           00395 ;
048F 0C05  00396      movlw    OPTION_CODE          ;SET UP PRESCALER, WDT on 18msec.
0490 0002  00397      option                        ;Clock TMR0 every 64 instruc cycles.
           00398 ;
0491 0078  00399      clrf     FLAG2                ; Clear out flag bank 2.
0492 0079  00400      clrf     FLAG3                ; Clear out flag bank 3.
           00401 ;
0493 0CFF  00402      movlw    0ff                  ; Display FF at start up
0494 003C  00403      movwf    BYTE_1               ; first byte = FF
0495 003D  00404      movwf    BYTE_2               ; second = FF
0496 003E  00405      movwf    BYTE_3               ; third = FF
0497 003F  00406      movwf    BYTE_4               ; fourth byte = FF
           00407 ;
0498 0C83  00408      movlw    MSEC8                ;TMR0 = 8 mSEC
0499 0021  00409      movwf    TMR0                 ;       /
           00410 ;
049A 095E  00411      call     ResetIR              ; get the IR ready to receive
049B 05B8  00412      bsf      FLAG2,LAST_IR_STATE  ; preset the IR flag for a
           00413                                    ; RecordRTCCatLowToHiTransition
049C 04A4  00414      bcf      FSR,5                ; File page 1
049D 04C4  00415      bcf      FSR,6                ; /
           00416 ;
           00417 ;************ Main loop Starts here. *************
049E       00418 IRMain
           00419 ;
           00420 ;
049E       00421 IRInnerLoop
049E 0665  00422      btfsc    PORTA,IR             ; ?IR rcvr not receiving an IR burst?
049F 06B8  00423      btfsc    FLAG2,LAST_IR_STATE  ; was it receiving a burst last time?
```

```
04A0 02A2  00424      incf     PCL,F                ; Not either
04A1 095A  00425      call     RecordRTCC_atLowToHiTransition
           00426                                     ; Record the TMR0 value when the lo to
           00427                                     ; hi transition came from the receiver
           00428 ;
04A2 0765  00429      btfss    PORTA,IR             ; ?IR receiver receiving an IR burst?
04A3 07B8  00430      btfss    FLAG2,LAST_IR_STATE  ; was it not rcv'g a burst last time?
04A4 02A2  00431      incf     PCL,f                ; Not either
04A5 0922  00432      call     ReadReceiver         ; read the new information
           00433
04A6 0C83  00434      movlw    MSEC8                ; Check for overflow.
04A7 0081  00435      subwf    TMR0,W               ; SEE IF TMR0 < MSEC8,
04A8 0603  00436      btfsc    STATUS,C             ; If TMR0 < MSEC8, Overflow.
04A9 0A9E  00437      goto     IRInnerLoop          ; No overflow, no carry, loop.
04AA 0900  00438      call     SvcTimer             ; Keep time and reload time keeper.
           00439 ;
04AB 04A3  00440      bcf      STATUS,PA0           ; get ready to call from page 1
04AC 04C3  00441      bcf      STATUS,PA1           ;  /
04AD 0912  00442      call     UpdateDisplay        ; rotate power to next display digit
           00443 ;
           00444  if BeginIr6121==200              ; return the bits to this page
           00445      bsf      STATUS,PA0           ; page 1
           00446      bcf      STATUS,PA1           ; /
           00447  endif
           00448  if BeginIr6121==400
04AE 04A3  00449      bcf      STATUS,PA0           ; page 2
04AF 05C3  00450      bsf      STATUS,PA1           ; /
           00451  endif
           00452  if BeginIr6121==600
           00453      bsf      STATUS,PA0           ; page 3
           00454      bsf      STATUS,PA1           ; /
           00455  endif
           00456 ;
04B0 07C6  00457      btfss     PORTB,RIGHT_OFF     ; Is display ready to display HOLD?
04B1 0778  00458      btfss     FLAG2,HOLD          ; IS the hold active?
04B2 0AB4  00459      goto      NotHold             ; do not turn on lite for HOLD indicate
04B3 04E7  00460      bcf       PORTC,DP            ; TURN on LED flag, show HOLD is active
04B4       00461 NotHold
           00462 ;
04B4 0639  00463      btfsc     FLAG3,EIGTH_SEC     ; check for 1/8 second time out
04B5 0963  00464      call      EigthSecChores      ; all that needs doing every 1/8 second
           00465                                     ; can go in this subroutine
           00466 ;
04B6 0619  00467      btfsc     FLAG3,_4TH_SEC      ; check for 1/4 second time out
04B7 097E  00468      call      QuarterSecChores    ; all that needs doing every 1/4sec
           00469                                     ; can go in this subroutine
           00470 ;
04B8 0659  00471      btfsc     FLAG3,TWO_SEC       ; check for two second time out
04B9 0985  00472      call      TwoSecChores        ; all that needs doing every two secs
           00473                                     ; can go in this subroutine
           00474
04BA 0A9E  00475      goto      IRMain
           00476 ;
0600       00199      org       600
0600       00200 BeginTeknika
           00201      include         "teknika.asm"
           00001           TITLE        "IR-Technica TV format Remote Control Detector V0.01"
           00002           SUBTITL      "Comments documentation and history"
           00003 ;
           00004 ;*************************************************************************
           00005 ; File Name :      TEKNIKA.ASM
           00006 ;*************************************************************************
           00007 ;      Author:    William G. Grimm
           00008 ;      Company:   Microchip Technology
           00009 ;      Revision:  V0.01
           00010 ;      Date:      March 31, 1996
```


© 1997 Microchip Technology Inc.

```
00011 ;      Assembler: MPASM version 1.21
00012 ;
00013 ;*******************************************************************************
00014 ;   Revision History:
00015 ;
00016 ;
00017 ;   V0.01    Original  March 28, 1996
00018 ;
00019 ;   V0.02    repaired bug that kept HOLD from operating
00020 ;            March 31, 1996
00021 ;
00022 ;   V0.03    modified gpa and pulse length subtraction
00023 ;            March 31, 1996
00024 ;
00025 ;*******************************************************************************
00026 ;   OPTION_CODE   EQU    B'00000101'     ;SET UP PRESCALER, WDT on 18msec.
00027 ;               Same as IR6121
00028 ;*******************************************************************************
00029 ;
00030 ;*******************************************************************************
00031 ; file memory location definitions
00032 ;*******************************************************************************
00033 ;
00034 ;  full byte file memory locations
00035 ;            (those commented out are defined in IR6121 or IRMAIN)
00036 ;
00037 ;    TIMER          EQU   0b    ; Bit5 = 1/4 second, Bit1 = 16 millisecs.
0000000C 00038     BUTTON         EQU   0c    ; holds last value of last button pressed
00039 ;
00040
00041 ;    RIGHT_DIGIT    EQU   0f        ; defined in irmain
00042 ;    READ_LH        EQU   11        ; Low to high reading is stored here.
00043 ;    IR_STATE       EQU   12        ; Which bit is coming in.
00044 ;    IR_BYTE13      EQU   13        ; First byte for collecting inputs.
00045 ;    IR_BYTE24      EQU   14        ; Second byte for collecting inputs.
00046 ;
00047 ;    FLAG2          EQU   18        ; flag bank 2
00048 ;    FLAG3          EQU   19        ; flag bank 3
00049 ;
00050 ;DEFINE FLAG2 REG FUNCTION:
00051 ;       BIT # 7|6|5|4|3|2|1|0|
00052 ;--------------|-|-|-|-|-|-|-|-|
00053 ;              | | | | | | |Y| --> Command Ready.
00054 ;              | | | | | |Y| | --> Command in process.
00055 ;              | | | | |Y| | | --> Most Significant bit of time stamp.
00056 ;              | | | |Y| | | | --> HOLD is active
00057 ;              | | |Y| | | | | --> 4 bytes have been rcv'd successfully
00058 ;              | |Y| | | | | | --> Value of last IR bit received.
00059 ;              |Y| | | | | | | --> A Valid hold received < 1/4 sec ago.
00060 ;            Y| | | | | | | | -->
00061 ; Y = DEFINED AS SHOWN (0/1)
00062 ;   (commented definitions are defined elsewhere)
00063 ;HOLD_RCVD     EQU 6
00064 ;LAST_IR_STATE EQU 5
00065 ;KEY_READY     EQU 4
00066 ;HOLD          EQU 3
00067 ;STAMP_MSB     EQU 2
00068 ;CMD_PEND      EQU 1               ; A channel command is pending.
00069 ;CMD_RDY       EQU 0               ; A channel command is ready.
00070 ;
00071 ;DEFINE FLAG3 REG FUNCTION:
00072 ;       BIT # 7|6|5|4|3|2|1|0|
00073 ;--------------|-|-|-|-|-|-|-|-|
00074 ;              | | | | | | |Y| --> Quarter second flag.
00075 ;              | | | | | |Y| | --> Eighth second flag.
00076 ;              | | | | |Y| | | --> Two second flag.
```

```
                00077 ;               | | | |Y| | | | --> A channel up has been received
                00078 ;               | | |Y| | | | | --> A Channel Down has been received
                00079 ;               | |Y| | | | | | -->
                00080 ;               |Y| | | | | | | -->
                00081 ;              Y| | | | | | | | -->
                00082 ; Y = DEFINED AS SHOWN (0/1)
                00083 ;   (commented definitions are defined elsewhere)
                00084 ;
                00085 ;_4TH_SEC       EQU 0
                00086 ;EIGTH_SEC      EQU 1
                00087 ;TWO_SEC        EQU 2
00000003        00088 ACTIVE_UP       EQU 3
00000004        00089 ACTIVE_DOWN     EQU 4
                00090 ;
                00091 ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                00092          SUBTITL "Constant definitions."
                00093 ;*************************************************************************
                00094 ; Gap length IR decoding time constants.  Values were derived from
                00095 ; successive readings made with MEASURE.ASM
                00096 ;  (commented defines are characterisitic of all 6121 remotes
                00097 ;   and are defined in IR6121)
                00098 ;HOLD_MIN      EQU  1a         ; Changed from 1e to minimize intermittency.
                00099 ;HOLD_MAX      EQU  27         ; Changed from 23 to minimize intermittency.
                00100 ;HEAD_MIN      EQU  38
                00101 ;HEAD_MAX      EQU  48
                00102 ;ONE_MAX       EQU  1f
                00103 ;ZERO_MAX      EQU  0e
                00104 ;
                00105 ;
                00106 ;  Definitions characteristic of Teknika TV remote controls
00000014        00107 ID_BYTE_1       EQU  14                ; Teknika signature byte 1
000000EB        00108 ID_BYTE_2       EQU  0eb               ; Teknika signature byte 2
                00109 ;
00000000        00110 TEK_ZERO        EQU  000               ; #0
00000001        00111 TEK_ONE         EQU  001               ; #1
00000002        00112 TEK_TWO         EQU  002               ; #2
00000003        00113 TEK_THREE       EQU  003               ; #3
00000008        00114 TEK_FOUR        EQU  008               ; #4
00000009        00115 TEK_FIVE        EQU  009               ; #5
0000000A        00116 TEK_SIX         EQU  00a               ; #6
0000000B        00117 TEK_SEVEN       EQU  00b               ; #7
00000010        00118 TEK_EIGHT       EQU  010               ; #8
00000011        00119 TEK_NINE        EQU  011               ; #9
0000001B        00120 TEK_ON_OFF      EQU  01b               ; ON/OFF
0000001A        00121 TEK_MUTE        EQU  01a               ; Mute button
00000013        00122 TEK_CHUP        EQU  013               ; CHUP Clockwise
00000012        00123 TEK_CHDN        EQU  012               ; CHDN Counter Clockwise
                00124 ;
                00125 ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                00126          SUBTITL "Timer Routines."
                00127 ;*************************************************************************
                00128 ; Timer servicing routine
                00129 ;    Called every 8 milliseconds, this clears the
                00130 ;    watch dog, reloads TMR0
                00131 ;    and keeps track of relative time.
                00132 ;
0600            00133 TekServiceTimer
0600 0C83       00134       movlw     MSEC8              ;TMR0 = 8 milliseconds.
0601 01C1       00135       addwf     TMR0,W             ; Add overflow amount.
0602 0021       00136       movwf     TMR0               ;       /
0603 0004       00137       clrwdt
0604 03EB       00138       incfsz    TIMER,F            ; Increment the timer, Skip to two second
                00139                                    ; set up if it rolls over.
0605 0A08       00140       goto      TekCheckMatch      ; Go to other possible setups.
0606 0559       00141       bsf       FLAG3,TWO_SEC      ; Set the two second flag.
                00142 ;
```


© 1997 Microchip Technology Inc.

```
0607 0800  00143      retlw    0                ; sync serviced.
0608       00144 TekCheckMatch                  ; 1/8 and 1/4 second flags are staggered
           00145                                ; make more efficient use of processor time
0608 020B  00146      movf     TIMER,W          ; Bring in the timer.
0609 0E0F  00147      andlw    b'00001111'      ; Check lower bits.
060A 0F02  00148      xorlw    d'2'             ; 1/8sec chores called when low nibble=2.
060B 0643  00149      btfsc    STATUS,Z         ; Was the low nibble not 2?
060C 0539  00150      bsf      FLAG3,EIGTH_SEC  ; No! it was 2, Set the 1/8 sec flag.
060D 020B  00151      movf     TIMER,W          ; Bring in the timer.
060E 0E1F  00152      andlw    b'00011111'      ; Check five lower bits.
060F 0F19  00153      xorlw    0x19             ; 1/4 second chores called every 0x19.
0610 0643  00154      btfsc    STATUS,Z         ; Was the low five bits not 0x19?
0611 0519  00155      bsf      FLAG3,_4TH_SEC   ; No! it was 0x19, Set the 1/4 sec flag.
0612 0800  00156      retlw    0                ; matches checked, return.
           00157 ;
           00158 ;
           00159 ;
           00160 ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
           00161          SUBTITL "6121 type IR remote control reader."
           00162 ; ****************************************
           00163 ; The following reads the IR transmitter.
           00164 ; When the IR transmitter is being read,
           00165 ; this routine takes control of the clocks
           00166 ; and suspends all other functions.
           00167 ;
           00168 ; ****************************************
           00169 ; IR Receiver routine
           00170 ;
           00171 ;
           00172 ;-------------------------------------------------------------------------
           00173 ; TekRdAddr
           00174 ;   This routine determines whether a Teknika remote control is
           00175 ;   sending the command.
           00176 ;-------------------------------------------------------------------------
0613       00177 TekRdAddr
0613 0C14  00178      movlw    ID_BYTE_1        ; bring in what would be the first byte
0614 0193  00179      xorwf    IR_BYTE13,W      ; compare with the byte that came in first
0615 0743  00180      btfss    STATUS,Z         ; if the same, the zero bit is set
0616 0A82  00181      goto     TekIRReset       ; wrong ID, reset the IR state
0617 0CEB  00182      movlw    ID_BYTE_2        ; bring in what would be the second byte
0618 0194  00183      xorwf    IR_BYTE24,W      ; compare with the byte that came in 2nd
0619 0743  00184      btfss    STATUS,Z         ; if the same, the zero bit is set
061A 0A82  00185      goto     TekIRReset       ; wrong ID, reset the IR state
061B 0800  00186      retlw    0                ; Success! A Teknika remote sent the command
           00187 ;
           00188 ;-------------------------------------------------------------------------
           00189 ; RD_COMMAND
           00190 ;   This routine determines what kind of command was sent, and places
           00191 ;   the result in BUTTON or one of the channel up or down flags.
           00192 ;-------------------------------------------------------------------------
061C       00193 TekRdCommand
061C 0253  00194      comf     IR_BYTE13,W      ; Byte3 must be the complement of Byte4
061D 0194  00195      xorwf    IR_BYTE24,W      ; For a valid command.
061E 0743  00196      btfss    STATUS,Z         ; If the same, skip to continue.
061F 0A82  00197      goto     TekIRReset       ; not complements, not valid.
           00198 ;
0620 0C13  00199      movlw    TEK_CHUP         ; check for a channel up command
0621 0193  00200      xorwf    IR_BYTE13,W      ; compare with byte 3
0622 0643  00201      btfsc    STATUS,Z         ; skip if no match
0623 0579  00202      bsf      FLAG3,ACTIVE_UP  ; active channel up
0624 0643  00203      btfsc    STATUS,Z         ; skip again if no active channel up
0625 0A39  00204      goto     TekLogCommand
           00205 ;
0626 0C12  00206      movlw    TEK_CHDN         ; check for a channel down command
0627 0193  00207      xorwf    IR_BYTE13,W      ; compare with byte 3
0628 0643  00208      btfsc    STATUS,Z         ; skip if no match
```

```
0629 0599  00209       bsf      FLAG3,ACTIVE_DOWN   ; active channel down
062A 0643  00210       btfsc    STATUS,Z            ; skip again if no active channel down
062B 0A39  00211       goto     TekLogCommand
           00212 ;
062C 0C0A  00213       movlw    d'10'               ; Look for ten possible buttons.
062D 0034  00214       movwf    IR_BYTE24           ; IR_BYTE24 is converted for use as
           00215                                     ; a counter.
062E       00216 GuessLoop
062E 00F4  00217       decf     IR_BYTE24,F         ; decrement to next button to look for
062F 0254  00218       comf     IR_BYTE24,W         ; See if it rolled over to FF
0630 0643  00219       btfsc    STATUS,Z            ; If rollover, this not a valid command.
0631 0A39  00220       goto     TekLogCommand       ; not a listed button return.
0632 0214  00221       movf     IR_BYTE24,W         ; Bring in the counter, which is a
           00222                                     ; guess as to what the button is.
0633 093B  00223       call     TekTable            ; NO! Get the code for guessed value
0634       00224 CheckGuess                          ; for dog biscuit
0634 0193  00225       xorwf    IR_BYTE13,W         ; look for a match with the guess and
           00226                                     ; actual value which is in IR_BYTE13.
0635 0743  00227       btfss    STATUS,Z            ; If it matchs skip and stop looping.
0636 0A2E  00228       goto     GuessLoop           ; No match.  Guess again.
0637       00229 TekLogButton
0637 0214  00230       movf     IR_BYTE24,W         ; bring in the count
0638 002C  00231       movwf    BUTTON              ; it has the new button number
0639       00232 TekLogCommand
0639 0598  00233       bsf      FLAG2,KEY_READY     ; Good set received
063A 0A6C  00234       goto     TekLogHold          ; Activate the hold for the first pass.
           00235 ;
063B       00236 TekTable
063B 01E2  00237       addwf    PCL,F               ; Computed jump for look-up table.
063C 0800  00238       retlw    TEK_ZERO            ; #0
063D 0801  00239       retlw    TEK_ONE             ; #1
063E 0802  00240       retlw    TEK_TWO             ; #2
063F 0803  00241       retlw    TEK_THREE           ; #3
0640 0808  00242       retlw    TEK_FOUR            ; #4
0641 0809  00243       retlw    TEK_FIVE            ; #5
0642 080A  00244       retlw    TEK_SIX             ; #6
0643 080B  00245       retlw    TEK_SEVEN           ; #7
0644 0810  00246       retlw    TEK_EIGHT           ; #8
0645 0811  00247       retlw    TEK_NINE            ; #9
           00248 ;
           00249 ;
           00250 ;--------------------------------------------------------------------------
           00251 ; TekRdRcvr
           00252 ;  Second part of the IR receiver.  It takes the present count of
           00253 ;  TMR0 and subtracts the count recorded when the receiver output
           00254 ;  went high (READ_LH) to find the dark pulse duration.  In that duration
           00255 ;  will be encoded the 1, 0, HOLD, or attention.
           00256 ;--------------------------------------------------------------------------
0646       00257 TekRdRcvr
0646 04B8  00258       bcf      FLAG2,LAST_IR_STATE ; Record that the IR rcvr output
           00259                                     ; is now high
           00260                                     ; Calculate length of the dark pulse,
           00261                                     ; length of time receiver was high.
           00262                                     ; (placed in READ_LH)
0647 0211  00263       movf     READ_LH,W           ; bring in the start measurement
0648 0081  00264       subwf    TMR0,W              ; subtract the final from the start
0649 0031  00265       movwf    READ_LH             ; gap or pulse length is now in
           00266                                     ; READ_LH, must be checked
064A 0C83  00267       movlw    MSEC8               ; Base number of TMR0 count.
064B 0091  00268       subwf    READ_LH,W           ; Subtract the base count of TMR0
064C 0603  00269       btfsc    STATUS,C            ; skip the store and toss value if neg
064D 0031  00270       movwf    READ_LH             ; value was positive, store
           00271 ;
064E       00272 TekMathDone
           00273 ;
064E 0678  00274       btfsc    FLAG2,HOLD          ; is it now looking for holds?
```


© 1997 Microchip Technology Inc.

```
064F 0A64  00275       goto     TekLookHold          ; look for HOLD
           00276 ;
0650 096F  00277       call     TekLookAtten         ; look for an attention dark pulse
0651 01E2  00278       addwf    PCL,F                ; skip if 1 was ret'd, no atten pulse
0652 0800  00279       retlw    0                    ; a 0 ret'd, ATTEN pulse found, return
           00280 ;
0653 0C1F  00281       movlw    ONE_MAX              ; Test for the max length of one.
0654 0091  00282       subwf    READ_LH,W            ; If no carry is gen'd, get a valid one
0655 0603  00283       btfsc    STATUS,C             ; No carry means the read is below max
0656 0A82  00284       goto     TekIRReset           ; IR no good, Above maximum is invalid.
           00285 ;
0657 0C0E  00286       movlw    ZERO_MAX             ; Test for the max length of Zero.
0658 0091  00287       subwf    READ_LH,W            ; If no carry is gen'd, get a valid 0.
           00288                                     ; the carry now has the newly rcv'd bit
           00289                                     ; shift the bit into the proper location
           00290 ;
0659 0772  00291       btfss    IR_STATE,3           ; Every 8 states gives dest changes
065A 0333  00292       rrf      IR_BYTE13,F          ; this bit is a part of IR byte 1 or 3
065B 0672  00293       btfsc    IR_STATE,3           ;   /
065C 0334  00294       rrf      IR_BYTE24,F          ; this bit is a part of ir byte 2 or 4
           00295 ;
065D 0C01  00296       movlw    1                    ; Get ready to add one to the IR STATE
065E 01F2  00297       addwf    IR_STATE,F           ; inc the state setting half carry bits
065F 0723  00298       btfss    STATUS,DC            ; skip if digit carry generated
0660 0800  00299       retlw    0                    ; all done reading for now
0661 07B2  00300       btfss    IR_STATE,5           ; check to determine if the 1st and 2nd
           00301                                     ; bytes or 3rd and 4th bytes now ready
0662 0A13  00302       goto     TekRdAddr            ; First and second byte ready.
0663 0A1C  00303       goto     TekRdCommand         ; Third and fourth byte ready.
           00304 ;
           00305 ;-------------------------------------------------------------------------
           00306 ; LOOK_HOLD
           00307 ;  Reads the length of the received dark pulse and determines if
           00308 ;  a valid HOLD pulse has been received
           00309 ;-------------------------------------------------------------------------
0664       00310 TekLookHold
0664 0C1A  00311       movlw    HOLD_MIN             ; Find if between hold and one.
0665 0091  00312       subwf    READ_LH,W            ; IF no carry is gen'd, The read is between
0666 0703  00313       btfss    STATUS,C             ; HOLD and one and as such, invalid.
0667 0800  00314       retlw    0                    ; Ret to main routine from invalid read.
0668 0C27  00315       movlw    HOLD_MAX             ; Test for the max length of HOLD.
0669 0091  00316       subwf    READ_LH,W            ; If no carry is gen'd, get a valid hold.
066A 0603  00317       btfsc    STATUS,C             ;
066B 0800  00318       retlw    0
066C       00319 TekLogHold
066C 0578  00320       bsf      FLAG2,HOLD           ; valid HOLD received
066D 05D8  00321       bsf      FLAG2,HOLD_RCVD      ; clear bit for the next hold condition
066E 0800  00322       retlw    0
           00323 ;
           00324 ;-------------------------------------------------------------------------
           00325 ; LOOK_ATTEN
           00326 ;  Reads the length of the received dark pulse and determines if
           00327 ;  a valid attention pulse has been received
           00328 ;-------------------------------------------------------------------------
066F       00329 TekLookAtten                      ; look for attention dark pulse
066F 0C38  00330       movlw    HEAD_MIN             ; Find if between head and one.
0670 0091  00331       subwf    READ_LH,W            ; IF no carry is gen'd, Reading between
0671 0703  00332       btfss    STATUS,C             ; HOLD and HEAD and as such, invalid.
0672 0A79  00333       goto     TekCheckState        ; continue, no attention gap.
0673 0C48  00334       movlw    HEAD_MAX             ; Test for the max length of HEAD.
0674 0091  00335       subwf    READ_LH,W            ; If no carry is gen'd, get a valid head
0675 0603  00336       btfsc    STATUS,C             ; A carry = a too long gap = invalid.
0676 0A79  00337       goto     TekCheckState        ; continue, no attention gap
0677 0072  00338       clrf     IR_STATE             ; Valid Atten dark pulse. Command starts
           00339                                     ; the state machine looking for bits.
0678 0800  00340       retlw    0                    ; return to main routine, ATTEN found
```


© 1997 Microchip Technology Inc.

```
0679        00341 TekCheckState
0679 0CE0   00342       movlw   0e0                 ; load A mask to mask all count states
067A 0152   00343       andwf   IR_STATE,W          ; compare with present state
067B 0743   00344       btfss   STATUS,Z
067C 0800   00345       retlw   0                   ; not a count state, ret to main routine
067D 0801   00346       retlw   01                  ; counting state, look for 1's and 0's
            00347 ;
            00348 ;-------------------------------------------------------------------------
            00349 ; RECORD_LH
            00350 ;  First part of the IR receiver.  It records the time when the
            00351 ;  output of the IR receiver went from low to high.  this creates the
            00352 ;  starting time for timing an IR pulse.
            00353 ;-------------------------------------------------------------------------
067E        00354 TekRecordLH
067E 05B8   00355       bsf     FLAG2,LAST_IR_STATE ; record that IR was last in dark pulse
067F 0201   00356       movf    TMR0,W              ; bring in the clock time
0680 0031   00357       movwf   READ_LH             ; record for when it goes back low
0681 0800   00358       retlw   0
            00359 ;-------------------------------------------------------------------------
            00360 ; TekIRReset
            00361 ;  Resets the IR state machine to ready it for receiving IR messages.
            00362 ;-------------------------------------------------------------------------
0682        00363 TekIRReset
0682 0478   00364       bcf     FLAG2,HOLD          ; not seen clear the hold
0683 04D8   00365       bcf     FLAG2,HOLD_RCVD     ; clear bit for the next hold condition
0684 0479   00366       bcf     FLAG3,ACTIVE_UP     ; clear channel up if present
0685 0499   00367       bcf     FLAG3,ACTIVE_DOWN   ; clear channel down if present
0686 0072   00368       clrf    IR_STATE            ; preset IR_STATE to -1
0687 0272   00369       comf    IR_STATE,F          ;  /
0688 0800   00370       retlw   0
            00371 ;
            00372 ;-------------------------------------------------------------------------
            00373 ; TekSvcHold
            00374 ;  Uses the HOLD to increment or decrement the BUTTON number.
            00375 ;-------------------------------------------------------------------------
0689        00376 TekSvcHold
0689 0679   00377       btfsc   FLAG3,ACTIVE_UP     ; is Channel up now present?
068A 0A8E   00378       goto    IncButton           ; Yes, increment button
068B 0699   00379       btfsc   FLAG3,ACTIVE_DOWN   ; is Channel Down now present?
068C 0A94   00380       goto    DecButton           ; Yes, Decrement button
068D 0800   00381       retlw   0                   ; neither now active
068E        00382 IncButton
068E 02AC   00383       incf    BUTTON,F            ; increment button
068F 0C0A   00384       movlw   d'10'               ;
0690 008C   00385       subwf   BUTTON,W            ; Compare with 10
0691 0603   00386       btfsc   STATUS,C            ; is BUTTON < 10?
0692 006C   00387       clrf    BUTTON              ; No recycle
0693 0800   00388       retlw   0
0694        00389 DecButton
0694 00EC   00390       decf    BUTTON,F            ; Decrement button
0695 024C   00391       comf    BUTTON,W            ; Roll to FF?
0696 0743   00392       btfss   STATUS,Z            ; ship if roll over
0697 0800   00393       retlw   0
0698 0C09   00394       movlw   d'9'                ; recycle on zero
0699 002C   00395       movwf   BUTTON              ;  /
069A 0800   00396       retlw   0
            00397 ;
            00398 ;*************************************************************************
            00399 ; The following subroutines are called by the executive
            00400 ; every 1/8 second, every 1/4 second, and every two seconds
            00401 ;
069B        00402 TekEigthSec                       ; all that needs doing every 1/8 sec
            00403                                   ; can be placed in this subroutine
069B 0439   00404       bcf     FLAG3,EIGTH_SEC     ; clear the time out flag
            00405 ;
069C 0800   00406       retlw   0
```


© 1997 Microchip Technology Inc.

```
                00407 ;
069D            00408 TekQuarterSec                       ; all that needs doing every 1/4 second
                00409                                     ; can be placed in this subroutine
069D 0678       00410       btfsc   FLAG2,HOLD            ; Check for HOLD condition still valid
069E 06D8       00411       btfsc   FLAG2,HOLD_RCVD       ; Check to see if a hold pulse has been
                00412                                     ; seen in the last 1/4 second
069F 02A2       00413       incf    PCL,F
06A0 0982       00414       call    TekIRReset            ; reset the IR state machine and get
                00415                                     ; ready for next
06A1 04D8       00416       bcf     FLAG2,HOLD_RCVD       ; Clear the hold received flag, it is
                00417                                     ; to be set by IR controller
06A2 0678       00418       btfsc   FLAG2,HOLD            ; check for active hold
06A3 0989       00419       call    TekSvcHold            ; service the hold function
                00420 ;
06A4 0419       00421       bcf     FLAG3,_4TH_SEC        ; clear the 1/4 second time-out
06A5 0800       00422       retlw   0
                00423 ;
06A6            00424 TekTwoSec                           ; things done every two seconds
06A6 0C0F       00425       movlw   A_CONFIG              ; setup for PORTA in loop, so
                00426                                     ; microcontroller will never forget
06A7 0005       00427       tris    PORTA                 ; inputs are on bits 0,1, and 2.
06A8 0C3F       00428       movlw   B_CONFIG              ; PORTB inputs are not used,
06A9 0006       00429       tris    PORTB                 ; PORTB outputs control digit drives
06AA 0C00       00430       movlw   C_CONFIG2
06AB 0007       00431       tris    PORTC                 ; PORTC is all outputs
06AC 0498       00432       bcf     FLAG2,KEY_READY       ; Routine that would interpret the key
                00433                                     ; will clear the flag that says it is
                00434                                     ; ready
06AD 0459       00435       bcf     FLAG3,TWO_SEC         ; clear the two second time-out
06AE 0800       00436       retlw   0
                00437 ;
                00438 ;
                00439 ;*****************************************************************************
                00440 ;         Start HERE.
                00441 ;*****************************************************************************
06AF            00442 StartTek
06AF 09A6       00443       call    TekTwoSec             ; re-setup ports A and B
                00444 ;
06B0 0C05       00445       movlw   OPTION_CODE           ;SET UP PRESCALER, WDT on 18msec.
06B1 0002       00446       option                        ;Clock TMR0 every 64 inst cycles.
                00447 ;
06B2 0078       00448       clrf    FLAG2                 ; Clear out flag bank 2.
06B3 0079       00449       clrf    FLAG3                 ; Clear out flag bank 3.
06B4 006C       00450       clrf    BUTTON                ; Displays Zero on reset
                00451 ;
                00452 ;
06B5 0C83       00453       movlw   MSEC8                 ;TMR0 = 8 mSEC
06B6 0021       00454       movwf   TMR0                  ;       /
                00455 ;
06B7 0982       00456       call    TekIRReset            ; get the IR ready to receive
06B8 05B8       00457       bsf     FLAG2,LAST_IR_STATE   ; preset the IR flag for a RECORD_LH
                00458 ;
06B9 04A4       00459       bcf     FSR,5                 ; File page 1
06BA 04C4       00460       bcf     FSR,6                 ; /
                00461 ;
                00462 ;************ Main loop Starts here. *************
06BB            00463 TekMain
                00464 ;
                00465 ;
06BB            00466 TekInnerLoop
06BB 0665       00467       btfsc   PORTA,IR              ; ?IR receiver not rcv'g an IR burst?
06BC 06B8       00468       btfsc   FLAG2,LAST_IR_STATE   ; was it receiving a burst last time?
06BD 02A2       00469       incf    PCL,F                 ; Not either
06BE 097E       00470       call    TekRecordLH           ; Record the TMR0 value when the lo to
                00471                                     ; hi transition came from the receiver
                00472 ;
```

```
06BF 0765  00473      btfss    PORTA,IR             ; ?IR receiver receiving an IR burst?
06C0 07B8  00474      btfss    FLAG2,LAST_IR_STATE  ; was it not rcv'g a burst last time?
06C1 02A2  00475      incf     PCL,F                ; Not either
06C2 0946  00476      call     TekRdRcvr            ; read the new information
           00477
06C3 0C83  00478      movlw    MSEC8                ; Check for overflow.
06C4 0081  00479      subwf    TMR0,W               ; SEE IF TMR0 < MSEC8,
06C5 0603  00480      btfsc    STATUS,C             ; If TMR0 < MSEC8, Overflow.
06C6 0ABB  00481      goto     TekInnerLoop         ; No overflow, no carry, loop.
06C7 0900  00482      call     TekServiceTimer      ; Keep time and reload time keeper.
           00483 ;
06C8 04A3  00484      bcf      STATUS,PA0           ; get ready to call from page 1
06C9 04C3  00485      bcf      STATUS,PA1           ;  /
           00486 ;
06CA 020C  00487      movf     BUTTON,W             ; get the IR measurement to be disp'd
06CB 0900  00488      call     LookUpDigit
06CC 0027  00489      movwf    PORTC                ; display on Right digit
           00490 ;
           00491  if BeginTeknika==200             ; return the bits to this page
           00492      bsf      STATUS,PA0           ; page 1
           00493      bcf      STATUS,PA1           ; /
           00494  endif
           00495  if BeginTeknika==400
           00496      bcf      STATUS,PA0           ; page 2
           00497      bsf      STATUS,PA1           ; /
           00498  endif
           00499  if BeginTeknika==600
06CD 05A3  00500      bsf      STATUS,PA0           ; page 3
06CE 05C3  00501      bsf      STATUS,PA1           ; /
           00502  endif
           00503 ;
06CF 0678  00504      btfsc    FLAG2,HOLD           ; IS the hold active?
06D0 04E7  00505      bcf      PORTC,DP             ; TURN on LED flag to show HOLD active
           00506 ;
06D1 05E6  00507      bsf      PORTB,LEFT_OFF       ; turn off the left digit
06D2 04C6  00508      bcf      PORTB,RIGHT_OFF      ; turn on the right digit
           00509 ;
06D3 0639  00510      btfsc    FLAG3,EIGTH_SEC      ; check for 1/8 second time-out
06D4 099B  00511      call     TekEigthSec          ; all that needs doing every 1/8 sec
           00512                                   ; can go in this subroutine
           00513 ;
06D5 0619  00514      btfsc    FLAG3,_4TH_SEC       ; check for 1/4 second time-out
06D6 099D  00515      call     TekQuarterSec        ; all that needs doing every 1/4 sec
           00516                                   ; can go in this subroutine
           00517 ;
06D7 0659  00518      btfsc    FLAG3,TWO_SEC        ; check for two second time-out
06D8 09A6  00519      call     TekTwoSec            ; all that needs doing every two sec
           00520                                   ; can go in this subroutine
           00521
06D9 0ABB  00522       goto    TekMain
           00523 ;
0100       00202      org           100
           00203 ;
           00204 ;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
           00205         SUBTITL "Start"
           00206 ;
           00207 ;*************************************************************************
           00208 ;          Start HERE.
           00209 ;*************************************************************************
0100       00210 StartAll
0100 0C1E  00211      movlw    C_CONFIG1            ; configuration to read from PORTC
0101 0007  00212      tris     PORTC                ; configure PORTC to read the bits
0102 0000  00213      nop                           ; allow time for the inputs to settle
0103 0747  00214      btfss    PORTC,SW2            ; check to see if jumper is in #2
0104 0B08  00215      goto     TekOr6121            ; Indicates IR6121 or TEK is requested
           00216  if BeginMeasure==200
```


© 1997 Microchip Technology Inc.

```
0105 05A3  00217       bsf       STATUS,PA0          ; page 1
0106 04C3  00218       bcf       STATUS,PA1          ; /
           00219   endif
           00220   if BeginMeasure==400
           00221       bcf       STATUS,PA0          ; page 2
           00222       bsf       STATUS,PA1          ; /
           00223   endif
           00224   if BeginMeasure==600
           00225       bsf       STATUS,PA0          ; page 3
           00226       bsf       STATUS,PA1          ; /
           00227   endif
0107 0A61  00228       goto      StartMeasure        ; Start the IR measurement routine
0108       00229 TekOr6121
0108 0727  00230       btfss     PORTC,SW1           ; check SW1 to determine if Tek or 6121
0109 0B0D  00231       goto      Teknika             ; jumper in, Teknika
           00232   if BeginIr6121==200
           00233       bsf       STATUS,PA0          ; page 1
           00234       bcf       STATUS,PA1          ; /
           00235   endif
           00236   if BeginIr6121==400
010A 04A3  00237       bcf       STATUS,PA0          ; page 2
010B 05C3  00238       bsf       STATUS,PA1          ; /
           00239   endif
           00240   if BeginIr6121==600
           00241       bsf       STATUS,PA0          ; page 3
           00242       bsf       STATUS,PA1          ; /
           00243   endif
010C 0A8E  00244       goto      StartIr6121         ; Start the 6121 IR format decoder
010D       00245 Teknika
           00246   if BeginTeknika==200
           00247       bsf       STATUS,PA0          ; page 1
           00248       bcf       STATUS,PA1          ; /
           00249   endif
           00250   if BeginTeknika==400
           00251       bcf       STATUS,PA0          ; page 2
           00252       bsf       STATUS,PA1          ; /
           00253   endif
           00254   if BeginTeknika==600
010D 05A3  00255       bsf       STATUS,PA0          ; page 3
010E 05C3  00256       bsf       STATUS,PA1          ; /
           00257   endif
010F 0AAF  00258       goto      StartTek            ; Start the Technika Remote decoder
           00259 ;
           00260 ;  START Vector
07FF       00261       org       0x07ff
07FF 0B00  00262       goto      StartAll            ; start vector
           00263 ;
           00264       END
```


© 1997 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX X--------------- ----------------
0100 : XXXXXXXXXXXXXXXX ---------------- ---------------- ----------------
0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0280 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXX-- ----------------
0400 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0440 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0480 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX-----
0600 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0640 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0680 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
06C0 : XXXXXXXXXXXXXXXX XXXXXXXXXX------ ---------------- ----------------
07C0 : ---------------- ---------------- ---------------- ---------------X
All other memory blocks unused.
Program Memory Words Used:   629
Program Memory Words Free:  1419
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

MICROCHIP

SECTION 3
PIC16CXXX
APPLICATION NOTES

3

Using the Analog-to-Digital (A/D) Converter - AN546 3-1
Implementing Wake-up on Key Stroke - AN552 3-21
Software Implementation of I²C™ Bus Master - AN554 3-25
Software Implementation of Asynchronous Serial I/O - AN555 3-85
Four Channel Digital Voltmeter with Display and Keyboard - AN557 3-121
Using the PORTB Interrupt on Change as an External Interrupt - AN566 3-149
Use of the SSP Module in the I²C™ Multi-Master Environment - AN578 3-153
Using the 8-Bit Parallel Slave Port - AN579 3-169
Using Timer1 in Asynchronous Clock Mode - AN580 3-177
Low-Power Real-Time Clock - AN582 3-181
Interfacing PICmicro™ Microcontrollers to an LCD Module - AN587 3-205
A PC-Based Development Programmer for the PIC16C84 - AN589 3-237
Apple® Desktop Bus (ADB™) - AN591 3-243
Using the CCP Module(s) - AN594 3-277
Implementing Ultrasonic Ranging - AN597 3-301
Air Flow Control Using Fuzzy Logic - AN600 3-319
Resistance and Capacitance Meter Using a PIC16C622 - AN611 3-339
Digital Signal Processing with the PIC16C74 - AN616 3-373
PIC14C000 Calibration Parameters - AN621 3-405
PIC14C000 A/D Theory and Implementation - AN624 3-411
Lead-Acid Battery Charger Implementation Using PIC14C000 - AN626 3-421
Adaptive Differential Pulse Code Modulation using PICmicro™ Microcontrollers - AN643 3-451
Yet Another Clock Featuring the PIC16C924 - AN649 3-491
PWM, a Software Solution for the PIC16CXXX - AN654 3-523
In-Circuit Serial Programming of Calibration Parameters Using a PICmicro™ Microcontroller- AN656 3-535
LCD Fundamentals Using PIC16C92X Microcontrollers - AN658 3-557
Smart Battery Charger with SMBus Interface - AN667 3-579

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

MICROCHIP

AN546

Using the Analog-to-Digital (A/D) Converter

Authors: *Sumit Mitra,*
Stan D'Souza, and
Russ Cooper
Microchip Technology Inc.

INTRODUCTION

This application note is intended for PIC16C7X users with some degree of familiarity with analog system design. The various sections discuss the following topics:

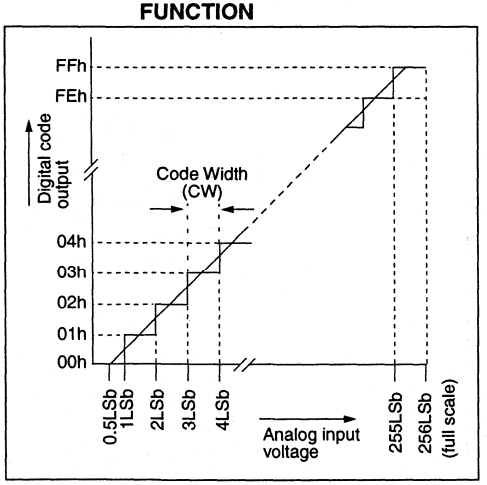
- Commonly used A/D terminology
- How to configure and use the PIC16C71 A/D
- Various ways to generate external reference voltage (VREF)
- Configuring the RA3:RA0 pins

COMMONLY USED A/D TERMINOLOGY

The Ideal Transfer Function

In an A/D converter, an analog voltage is mapped into an N-bit digital value. This mapping function is defined as the transfer function. An ideal transfer is one in which there are no errors or non-linearity. It describes the "ideal" or intended behavior of the A/D. Figure 1 shows the ideal transfer function for the PIC16C7X A/D.

FIGURE 1: PIC16C7X IDEAL TRANSFER FUNCTION



Note that the digital output value is 00h for the analog input voltage range of 0 to 1LSb. In some converters, the first transition point is at 0.5LSb and not at 1LSb as shown in Figure 2. Either way, by knowing the transfer function the user can appropriately interpret the data.

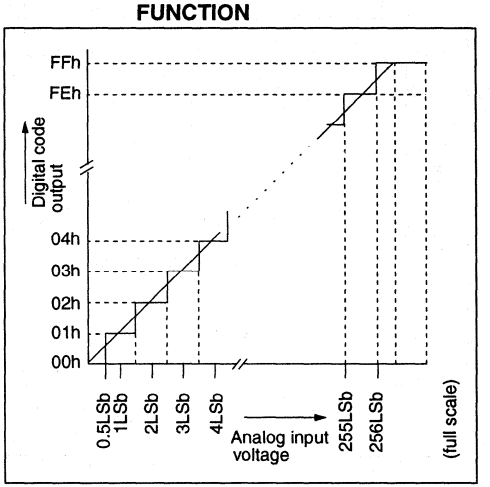
Transition Point

The analog input voltage at which the digital output switches from one code to the next is called the "Transition Point." The transition point is typically not a single threshold, but rather a small region of uncertainty (Figure 3). The transition point is therefore defined as the statistical average of many conversions. Stated differently, it is the voltage input at which the uncertainty of the conversion is 50%.

Code Width

The distance (voltage differential) between two transition points is called the "Code Width." Ideally the Code Width should be 1LSb (Figure 1).

FIGURE 2: ALTERNATE TRANSFER FUNCTION

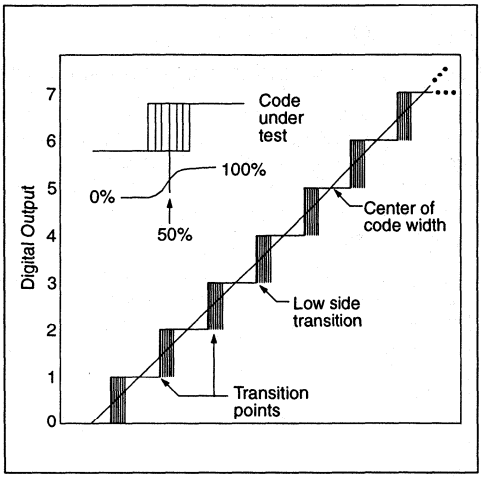


© 1997 Microchip Technology Inc.

Center of Code Width

The midpoint between two transition points is called the "Center of Code Width" (Figure 3).

FIGURE 3: TRANSITION POINTS



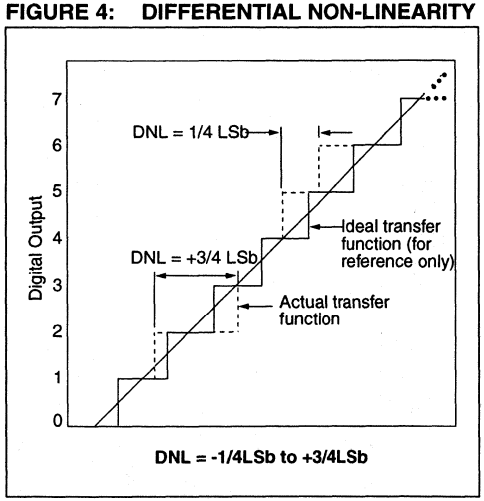
Differential Non-Linearity (DNL)

It is the deviation in code-width from 1LSb (Figure 4). The difference is calculated for each and every transition. The largest difference is reported as DNL.

It is important to note that the DNL is measured after the transfer function is normalized to match offset error and gain error.

Note that the DNL cannot be any less than -1LSb. In the other direction, DNL can be >1LSb.

FIGURE 4: DIFFERENTIAL NON-LINEARITY



Absolute Error

The maximum deviation between any transition point from the corresponding ideal transfer function is defined as the absolute error. This is how it is measured and reported in the PIC16C7X (Figure 5). The notable difference between absolute error and integral non-linearity (INL) is that the measured data is not normalized for full scale and offset errors in absolute error.

Absolute Error is probably the first parameter the user will review to evaluate an A/D. Sometimes absolute error is reported as the sum of offset, full-scale and integral non-linearity errors.

Total Unadjusted Error

Total Unadjusted Error is the same as absolute error. Again, sometimes it is reported as the sum of offset, full-scale and integral non-linearity errors.

No Missing Code

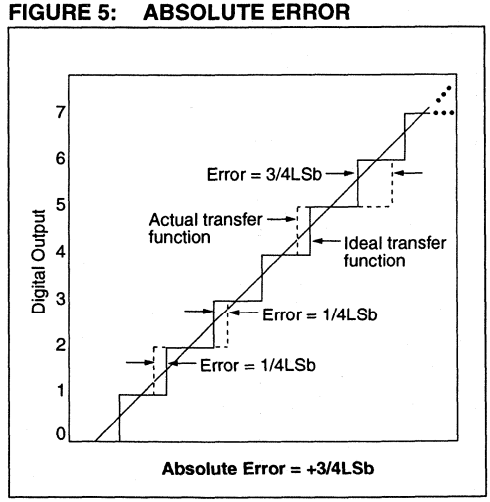
No missing code implies that as the analog input voltage is gradually increased from zero to full scale (or vice versa), all digital codes are produced. Stated otherwise, changing analog input voltage from one quantum of the analog range to the next adjacent range will not produce a change in the digital output by more than one code count.

Monotonic

Monotonicity guarantees that an increase (or decrease) in the analog input value will result in an equal or greater digital code (or less). Monotonicity does not guarantee that there are no missing codes. However, it is an important criterion for feedback control systems. Non-monotonicity may cause oscillations in such systems.

The first derivative of a monotonic function always has the same sign.

FIGURE 5: ABSOLUTE ERROR



© 1997 Microchip Technology Inc.

Ratiometric Conversion

Ratiometric Conversion is the A/D conversion process in which the binary result is a ratio of the supply voltage or reference voltage, the latter being equal to full-scale value by default. The PIC16C7X is a ratiometric A/D converter where the result depends on VDD or VREF.

In some A/Ds, an absolute reference is provided resulting in "absolute conversion".

Sample and Hold

In sample and hold type A/D converters, the analog input has a switch (typically a FET switch in CMOS) which is opened for a short duration to capture the analog input voltage onto an on-chip capacitor. Conversion is typically started after the sampling switch is closed.

Track and Hold

Track and Hold is basically the same as sample and hold, except the sampling switch is typically left on. Therefore the voltage on the on-chip holding capacitor "tracks" the analog input voltage. To begin a conversion, the sampling switch is closed.

The PIC16C7X A/D falls in this category.

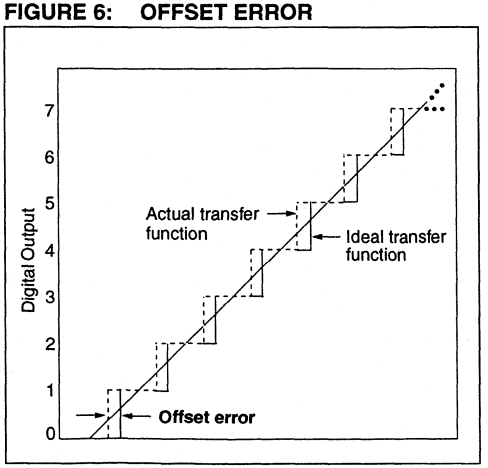
Sampling Time

Sampling Time is the time required to charge the on-chip holding capacitor to the same value as is on the analog input pin. The sampling time depends on the magnitude of the holding capacitor and the source impedance of the analog voltage input.

Offset Error (or Zero Error)

Offset Error is the difference between the first actual (measured) transition point and the first ideal transition point as shown in Figure 6. It can be corrected (by the user) by subtracting the offset error from each conversion result.

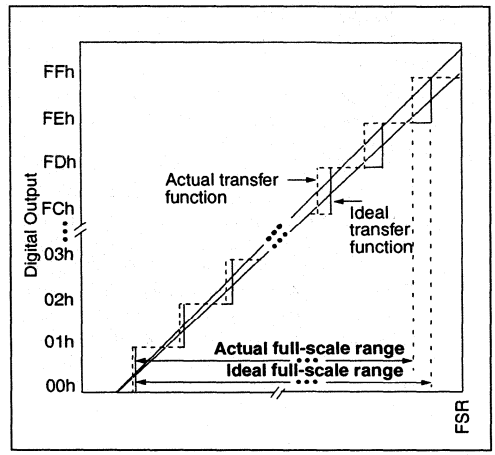
FIGURE 6: OFFSET ERROR



Full Scale Error (or Gain Error)

Full Scale Error is the difference between the ideal full scale and the actual (measured) full scale range (Figure 7). It is also called gain error, because the error changes the slope of the ideal transfer function creating a gain factor. It can be corrected (by the user) by multiplying each conversion result by the inverse of the gain.

FIGURE 7: FULL SCALE ERROR

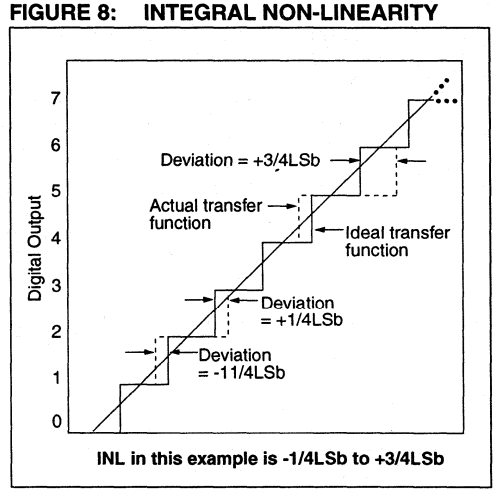


Integral Non-Linearity (INL), or Relative Error

The deviation of a transition point from its corresponding point on the ideal transfer curve is called "Integral Non-Linearity" (Figure 8). The maximum difference is reported as the INL of the converter.

It is important to note that Full Scale Error and the Offset Error are normalized to match end transition points before measuring the INL.

FIGURE 8: INTEGRAL NON-LINEARITY



HOW TO USE THE PIC16C71 A/D

The A/D in the PIC16C71 is easy to set up and use. There are a few considerations:

1. Select either VDD or VREF as reference voltage. (More on using VREF input later)

   Select A/D conversion clock (TAD): 2TOSC, 8TOSC, TOSC or TRC (internal RC clock). For the first three options, make sure that TAD ≥ 2.0 µs. If deterministic conversion time is required, select TOSC time-base. If conversion during SLEEP is required, select TRC.
2. Channel Selection: If only one A/D channel is required, program the ADCON1 register to 03h. This configures the A/D pins as digital I/O. If multiple channels are required, prior to each conversion the new channel must be selected.
3. Sampling and Conversion: After a new channel is selected, a minimum amount of sampling time must be allowed before the GO/$\overline{\text{DONE}}$ bit in ADCON0 is set to begin conversion. Once conversion begins, it is OK to select the next channel, ***but sampling does not begin until current conversion is complete***. Therefore, it is always necessary to ensure the minimum sampling time is provided for:
   i) after a conversion
   ii) after a new channel is selected
   iii) after A/D is turned on (bit ADON = 1)
4. Reading Result: Completion of a conversion can be determined by polling the GO/$\overline{\text{DONE}}$ bit (cleared), or polling flag bit ADIF (set), or waiting for an ADIF interrupt.

Additional tips:

a) Do not set bits GO/$\overline{\text{DONE}}$ and ADON in the same instruction. First, turn the A/D is on by setting bit ADON. Then allow at least 5 µs before conversion begins (setting the GO/$\overline{\text{DONE}}$ bit), longer if sampling time requirement is not met within 5 µs.

b) Aborting a conversion: A conversion can be aborted by clearing bit GO/$\overline{\text{DONE}}$. The A/D converter will stop conversion and revert back to sampling state.

c) Using the ADRES register as a normal register: The A/D only writes to the ADRES register at the end of a conversion. Therefore, it is possible to use the ADRES register as a normal file register between conversions and when A/D is off.

The following four examples provide sample code on using the A/D module.

EXAMPLE 1: HOW TO DO A SAMPLE A/D CONVERSION

```
;
;     InitializeAD, initializes and sets up the A/D hardware.
;     Always ch2, internal RC OSC.
InitializeAD
          bsf       STATUS, 5       ; select Bank1
          movlw     b'00000000'     ; select RA3-RA0
          movwf     ADCON1          ; as analog inputs
          bcf       STATUS, 5       ; select Bank0
          movlw     b'11010001'     ; select: RC osc, ch2...
          movwf     ADCON0          ; turn on A/D
Convert   call      sample-delay    ; provide necessary sampling time
;
          bsf       ADCON0, 2       ; start new A/D conversion
loop
          btfsc     ADCON0, 2       ; A/D over?
          goto      loop            ; no then loop
;
          movf      adres, w        ; yes then get A/D value
;
```

A detailed code listing is provided in Appendix A.

© 1997 Microchip Technology Inc.

EXAMPLE 2: SEQUENTIAL CHANNEL CONVERSIONS

```
;
;     InitializeAD, initializes and sets up the A/D hardware.
;     Select ch0 to ch3 in a round robin fashion, internal RC OSC.
;     Load results in 4 consecutive addresses starting at ADTABLE (10h)
;
InitializeAD
          bsf       STATUS, RP0       ; select Bank1
          movlw     b'00000000'       ; select RA3-RA0
          movwf     ADCON1            ; as analog inputs
          bcf       STATUS, RP0       ; select Bank0
          movlw     b'11000001'       ; select: RC osc, ch0...
          movwf     ADCON0            ; turn on A/D
          movlw     ADTABLE           ; point fsr to top of...
          movwf     FSR               ; table
;
new_ad    call      sample_delay      ; provide necessary sampling time
          bsf       ADCON0, GO        ; start new A/D conversion
loop
          btfsc     ADCON0, GO        ; A/D over?
          goto      loop              ; no then loop
;
          movf      adres, w          ; yes then get A/D value
          movwf     0                 ; load indirectly
          movlw     4                 ; select next channel
          addwf     ADCON0            ;     /
          bcf       ADCON0, ADIF      ; reset interrupt flag bit.
; increment pointer to correct table offset.
          clrf      temp              ; clear temp register
          btfsc     ADCON0, CH50      ; test lsb of channel select
          bsf       temp, 0           ; set if ch1 selected
          btfsc     ADCON0, CH51      ; test msb of channel select
          bsf       temp, 1           ;     /
          movlw     ADTABLE           ; get table address
          addwf     temp, w           ; add with temp
          movwf     FSR               ; move into indirect
          goto      new_ad
;
```

A detailed code listing is provided in Appendix B.

© 1997 Microchip Technology Inc.

EXAMPLE 3: SAMPLE INTERRUPT HANDLER FOR THE A/D

```
          org       0x00
          goto      start
          org       0x04
          goto      service_ad      ; interrupt vector
;
;         org       0x10
start
          movlw     b'00000000'     ;init I/O ports
          movwf     PORT_B
          tris      PORT_B
;
          call      InitializeAD
update
          bcf       flag, adover    ; reset software A/D flag
          call      SetupDelay      ; setup delay >= 10uS.
          bcf       ADCON0, adif    ; reset A/D int flag (ADIF
          bsf       ADCON0, go      ; start new A/D conversion
          bsf       INTCON, gie     ; enable global interrupt
loop
          btfsc     flag, adover    ; A/D over?
          goto      update          ; yes start new conv.
          goto      loop            ; no then keep checking
; InitializeAD, initializes and sets up the A/D hardware.
; select ch0 to ch3, RC OSC., a/d interrupt.
InitializeAD
          bsf       STATUS, RP0     ; select Bank1
          movlw     b'00000000'     ; select RA0-RA3...
          movwf     ADCON1          ; as analog inputs
          bcf       STATUS, RP0     ; select Bank0
          clrf      INTCON          ; clr all interrupts
          bsf       INTCON, ADIE    ; enable A/D int.
          movlw     b'11010001'     ; select: RC osc, ch2...
          movwf     ADCON0          ; turn on A/D
          return
;
service_ad
          btfss     ADCON0, ADIF    ; A/D interrupt?
          retfie                    ; no then ignore
          movf      ADRES, W        ; get A/D value
          return                    ; do not enable int
;
```

A detailed code listing is provided in Appendix C.

© 1997 Microchip Technology Inc.

EXAMPLE 4: CONVERSIONS DURING SLEEP MODE

```
;
;     InitializeAD, initializes and sets up the A/D hardware.
;     Select ch0 to ch3, internal RC OSC.
;     While doing the conversion put unit to sleep.  This will
;     minimize digital noise interference.
;     Note that A/D's RC osc. has to be selected in this instance.
;
InitializeAD
          bsf       STATUS, RP0     ; select Bank1
          movlw     b'00000000'     ; select RA0-RA3...
          movwf     ADCON1          ; as analog inputs
          bcf       STATUS, RP0     ; select Bank0
          movlw     b'11000001'     ; select: RC osc, ch0...
          movwf     ADCON0          ; turn on A/D & ADIE
          movlw     ADTABLE         ; point fsr to top of...
          movwf     FSR             ; table
;
new_ad
          bsf       ADCON0, GO      ; start new A/D conversion
          sleep                     ; goto sleep
; when A/D is over program will continue from here
;
          movf      ADRES, w        ; get A/D value
;
```

A detailed code listing is provided in Appendix D.

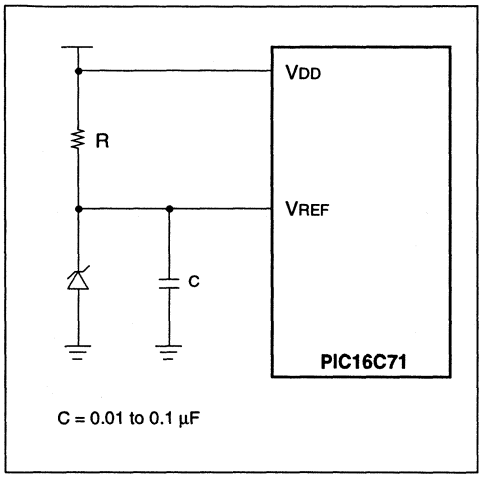
© 1997 Microchip Technology Inc.

USING EXTERNAL REFERENCE VOLTAGE

When using the external reference voltage, keep in mind that any analog input voltage must not exceed VREF.

An inexpensive way to generate VREF is by employing a zener diode (Figure 9). Most common zener diodes offer 5% accuracy. Reverse bias current may be as low as 10 µA. However, larger currents (1 mA - 20 mA) are recommended for stability, as well as lower impedance of the VREF source.

FIGURE 9: LOW COST VOLTAGE REFERENCE



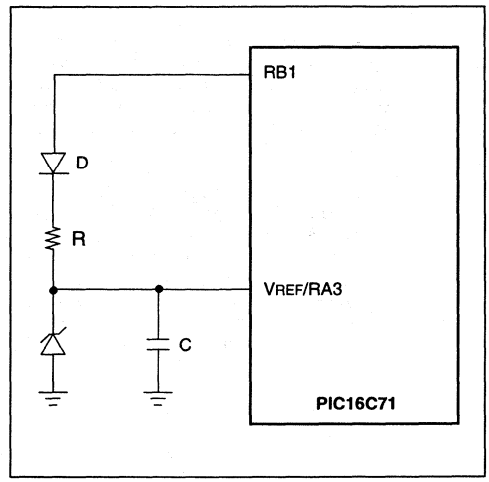
POWER MANAGEMENT IN USING VREF

In power sensitive applications, the user may turn on a VREF generator using another I/O pin (Figure 10). Drive a '1' on pin RB1, in this example, when using the A/D. Drive a '0' on pin RB1 when not using the A/D converter.

Note that this way RB1 is not floating. Even if VREF decays to some intermediate voltage, it will not cause the input buffer on RB1 to draw current.

Alternately, use RA0, RA1 or RA2 pin to supply the current instead of RB1. Configure the RA pin as analog (this will turn off its input buffer). Then use it as a digital output (Figure 11).

FIGURE 10: POWER-SENSITIVE APPLICATIONS #1



ZENERS AND REFERENCE GENERATORS

Finally, various reference voltage generator chips (typically using on-chip band-gap reference) are available. They are more accurate.

TABLE 1: ZENERS AND REFERENCE GENERATORS

Zeners	Vz	Tolerance
1N746	3.3V	±5%
1N747	3.6V	±5%
1N748	3.9V	±5%
1N749	4.3V	±5%
1N750	4.7V	±5%
1N751	5.1V	±5%
1N752	5.6V	±5%
Voltage Reference	**VREF**	**Tolerance**
AD580 (Maxim)	2.5V	±3% to ±0.4%
LM385	2.5V	±1.5%
LM1004	2.5V	±1.2%
LT1009 (LIN. Tech.)	2.5V	±0.2%
LT1019 (LIN. Tech.)	5.0V	±0.2%
LT1021 (LIN. Tech.)	5.0V	±0.05% to ±1%
LT1029 (LIN. Tech.)	5.0V	±0.2% to ±1%

© 1997 Microchip Technology Inc.

VREF IMPEDANCE AND CURRENT SUPPLY REQUIREMENTS

Ideally, VREF should have as low a source impedance as possible. Referring to Figure 9, VREF source impedence ≈ R. However, smaller R increases current consumption. Since VREF is used to charge capacitor arrays inside the A/D converter and the holding capacitor, Chold ≈ 51 pF, the following guideline should be met:

$$TAD = 6(1k + R)51.2pF + 1.677\mu s$$

TAD = conversion clock. For TAD = 2 µs and for CHOLD = 50 pF, VREF ≈ 50Ω.

For VREF impedance higher than this, the conversion clock (TAD) should be increased appropriately.

FIGURE 11: POWER-SENSITIVE APPLICATIONS #2

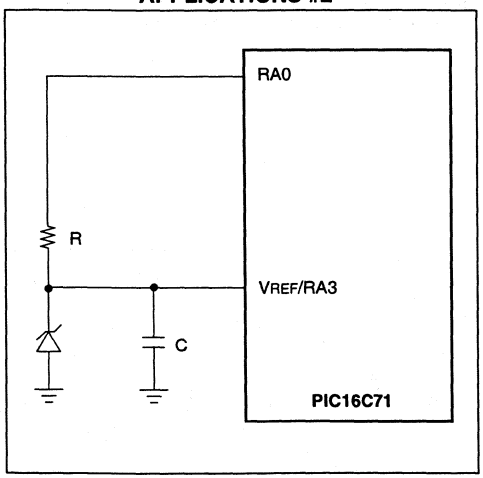


Table 2 gives examples of the maximum rate of conversion per bit, relating to the voltage reference impedance.

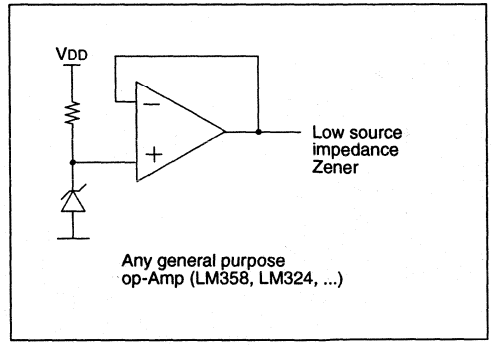
TABLE 2: MAXIMUM RATE OF CONVERSION / BIT

RVREF	TAD (Max)
1k	2.29 µs
5k	3.52 µs
10k	5.056 µs
50k	16.66 µs
100k	32.70 µs

Assumes no external capacitors

To achieve a low source impedance when using a Zener diode, a voltage follower circuit is recommended. This is shown in Figure 12.

FIGURE 12: VOLTAGE FOLLOWER CIRCUIT



CONFIGURING PORTA INPUTS AS ANALOG OR DIGITAL

Two bits in the ADCON1 register, PCFG1 and PCFG0, control how pins RA3:RA0 are configured.

When any of these pins are selected as analog:

- The digital input buffer is turned off to save current (Figure 13). Reading the port will read this pin as '0'.
- The TRIS bit still controls the output buffer on this pin. So, normally the TRIS bit will be set (input).
- However, if the TRIS bit is cleared, then the pin will output whatever is in the data latch.

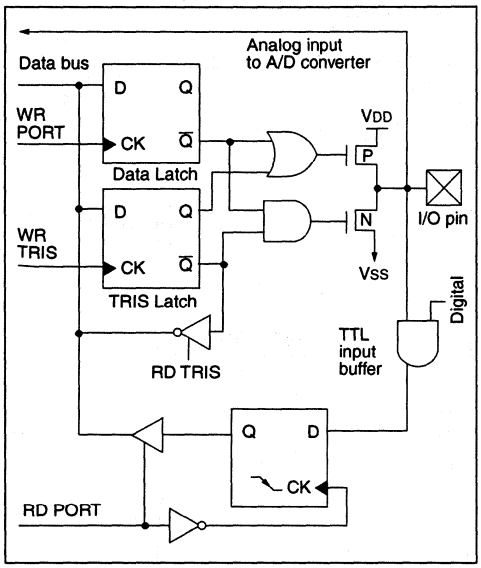
When any of these pins are selected as digital:

- The analog input still directly connects to the A/D and therefore the pin can be used as analog input.
- The digital input buffer is not disabled.

The user has, therefore, great flexibility in configuring these pins.

3

FIGURE 13: BLOCK DIAGRAM OF RA3:RA0 PINS



CURRENT CONSUMPTION THROUGH INPUT BUFFER

A CMOS input buffer will draw current when the input voltage is near its threshold (Figure 14).

In power-sensitive applications, the RA pins, when used as analog inputs, should be configured as "analog" to avoid unintended power drain.

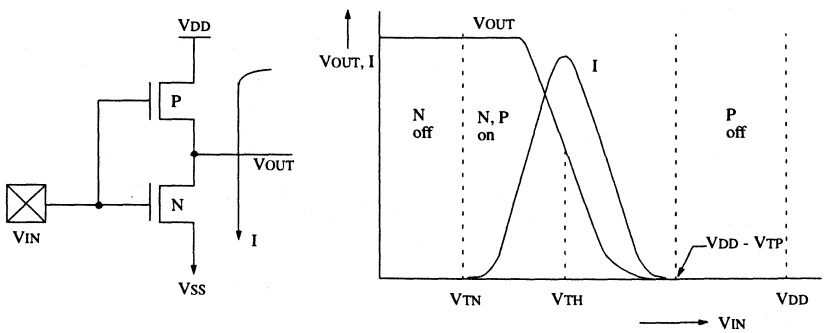
Other considerations and tips:

1. If possible, avoid any digital output next to analog inputs.
2. Avoid digital inputs that switch frequently (e.g., clocks) next to analog inputs.
3. If VREF is used, then ensure that no analog pin being sampled exceeds VREF.

SUMMARY

The PIC16C71 A/D converter is simple to use. It is versatile and has low power consumption.

FIGURE 14: A SIMPLE CMOS INPUT BUFFER



VTH = Threshold of the inverter

VTN = Device threshold of NMOS pull-down

-VTP = Device threshold of PMOS pull-up

I = On-current (or through current) of the inverter

IMAX = Maximum on-current occurs when VIN = VTH. Value of IMAX depends on the sizes of the devices. The larger the devices, the faster the input buffer, and the larger the value of IMAX. Typically, IMAX is 0.2 mA – 1 mA.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: SINGLE CHANNEL A/D (SAD)

```
MPASM 01.40 Released              SAD.ASM   1-16-1997  15:22:04         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;TITLE   "Single channel A/D (SAD)"
                     00002 ;This program is a simple implementation of the PIC16C71's
                     00003 ;A/D. 1 Channel is selected (CH0).
                     00004 ;The A/D is configured as follows:
                     00005 ;       Vref = +5V internal.
                     00006 ;       A/D Osc. =  internal RC
                     00007 ;       A/D Channel = CH0
                     00008 ;Hardware for this program is the PICDEM1 board.
                     00009 ;
                     00010 ;
                     00011 ;       Program:          SAD.ASM
                     00012 ;       Revision Date:
                     00013 ;                         1-14-97      Compatibility with MPASMWIN 1.40
                     00014 ;
                     00015 ;
                     00016         LIST P=16C71
                     00017         ERRORLEVEL  -302
                     00018 ;
                     00019         include "p16c71.inc"
                     00001         LIST
                     00002 ;P16C71.INC  Standard Header File, Version 1.00 Microchip Technology
                     00142         LIST
                     00020 ;
  00000010           00021 TEMP    EQU     10h
  00000001           00022 adif    equ     1
  00000002           00023 adgo    equ     2
                     00024 ;
0000                 00025         ORG     0x00
                     00026 ;
                     00027 ;
0000 2810            00028         goto    start
                     00029 ;
0004                 00030         org     0x04
0004 281E            00031         goto    service_int      ;interrupt vector
                     00032 ;
                     00033 ;
0010                 00034         org     0x10
0010                 00035 start
0010 3000            00036         movlw   B'00000000'      ;set port b as
0011 0086            00037         movwf   PORTB            ;all outputs
                     00038 ;       tris    PORTB            ;       /
0012 1683            00039         BSF     STATUS, RP0      ; Bank1
0013 0086            00040         MOVWF   TRISB            ; PortB as outputs
0014 1283            00041         BCF     STATUS, RP0      ; Bank0
                     00042 ;
0015 201F            00043         call    InitializeAD
0016                 00044 update
0016 0809            00045         movf    ADRES,W          ;get a/d value
0017 0086            00046         movwf   PORTB            ;output to port b
0018 2027            00047         call    SetupDelay       ;setup time >= 10uS.
0019 1088            00048         bcf     ADCON0,adif      ;clear int flag
001A 1508            00049         bsf     ADCON0,adgo      ;start new conversion
001B                 00050 loop
001B 1888            00051         btfsc   ADCON0,adif      ;a/d done?
001C 2816            00052         goto    update           ;yes then update new value.
```

```
001D 281B           00053         goto    loop            ;no then keep checking
                    00054 ;
                    00055 ;no interrupts are enabled, so if the program ever reaches here,
                    00056 ;it should be returned with the global interrupts disabled.
001E                00057 service_int
001E 0008           00058         return                  ;do not enable global.
                    00059 ;
                    00060 ;
                    00061 ;
                    00062 ;InitializeAD, initializes and sets up the A/D hardware.
                    00063 ;Select ch0 to ch3 as analog inputs, fosc/2 and read ch3.
                    00064 ;
001F                00065 InitializeAD
001F 1683           00066         bsf     STATUS,5        ;select Bank1
0020 3000           00067         movlw   B'00000000'     ;select ch0-ch3...
0021 0088           00068         movwf   ADCON1          ;as analog inputs
0022 1283           00069         bcf     STATUS,5        ;select Bank0
0023 30C1           00070         movlw   B'11000001'     ;select:RC,ch0..
0024 0088           00071         movwf   ADCON0          ;turn on A/D.
0025 0189           00072         clrf    ADRES           ;clr result reg.
0026 0008           00073         return
                    00074 ;
                    00075 ;This routine is a software delay of 10uS for the a/d setup.
                    00076 ;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
                    00077 ;a value of 3 to give 9uS, plus the move etc should result in
                    00078 ;a total time of > 10uS.
0027                00079 SetupDelay
0027 3003           00080         movlw   .3
0028 0090           00081         movwf   TEMP
0029                00082 SD
0029 0B90           00083         decfsz  TEMP, F
002A 2829           00084         goto    SD
002B 0008           00085         return
                    00086
                    00087
                    00088         END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---X----------- XXXXXXXXXXXXXXXX XXXXXXXXXXXX---- ----------------

All other memory blocks unused.

Program Memory Words Used:    30
Program Memory Words Free:   994


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     2 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B

```
MPASM 01.40 Released            SLPAD.ASM   1-16-1997  15:22:32         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001
                     00002 ;TITLE   "A/D in Sleep Mode"
                     00003 ;This program is a simple implementation of the PIC16C71's
                     00004 ;A/D feature. This program demonstrates
                     00005 ;how to do a a/d in sleep mode on the PIC16C71.
                     00006 ;The A/D is configured as follows:
                     00007 ;       Vref = +5V internal.
                     00008 ;       A/D Osc. = internal RC
                     00009 ;       A/D Interrupt = OFF
                     00010 ;       A/D Channels = ch 0
                     00011 ;
                     00012 ;The ch0 A/D result is displayed as a 8 bit binary value
                     00013 ;on 8 leds connected to port b. Hardware used is that of
                     00014 ;the PICDEMO board.
                     00015 ;
                     00016 ;
                     00017 ;       Program:          SLPAD.ASM
                     00018 ;       Revision Date:
                     00019 ;                         1-14-97      Compatibility with MPASMWIN 1.40
                     00020 ;
                     00021 ;
                     00022         LIST P=16C71
                     00023         ERRORLEVEL -302
                     00024 ;
                     00025         include "p16c71.inc"
                     00001         LIST
                     00002 ;P16C71.INC  Standard Header File, Version 1.00 Microchip Technology
                     00142         LIST
                     00026 ;
  00000010           00027 TEMP    EQU     10h
  00000001           00028 adif    equ     1
  00000002           00029 adgo    equ     2
                     00030 ;
                     00031 ;
0000                 00032         ORG     0x00
                     00033 ;
                     00034 ;
0000 2810            00035         goto    start
                     00036 ;
0004                 00037         org     0x04
0004 281D            00038         goto    service_int      ;interrupt vector
                     00039 ;
                     00040 ;
0010                 00041         org     0x10
0010                 00042 start
0010 3000            00043         movlw   B'00000000'      ;make port b all
0011 0086            00044         movwf   PORTB            ;outputs.
                     00045 ;       tris    PORTB            ;       /
0012 1683            00046         BSF     STATUS, RP0      ; Bank1
0013 0086            00047         MOVWF   TRISB            ; PortB as outputs
0014 1283            00048         BCF     STATUS, RP0      ; Bank0
                     00049 ;
0015 201E            00050         call    InitializeAD
0016                 00051 update
```

```
0016 0809           00052         movf    ADRES,W
0017 0086           00053         movwf   PORTB           ;save in table
0018 2027           00054         call    SetupDelay      ;
0019 1088           00055         bcf     ADCON0,adif     ;clr a/d flag
001A 1508           00056         bsf     ADCON0,adgo     ;start new a/d conversion
                    00057 ;
001B 0063           00058         sleep
001C 2816           00059         goto    update          ;wake up and update
                    00060 ;
001D                00061 service_int
001D 0008           00062         return                  ;do not enable int
                    00063 ;
                    00064 ;InitializeAD, initializes and sets up the A/D hardware.
001E                00065 InitializeAD
001E 1683           00066         bsf     STATUS,5        ;select Bank1
001F 3000           00067         movlw   B'00000000'     ;select ch0-ch3...
0020 0088           00068         movwf   ADCON1          ;as analog inputs
0021 1283           00069         bcf     STATUS,5        ;select Bank0
0022 30C1           00070         movlw   B'11000001'     ;select:internal RC, ch0.
0023 0088           00071         movwf   ADCON0          ;turn on a/d
0024 018B           00072         clrf    INTCON          ;clear all interrupts
0025 170B           00073         bsf     INTCON,ADIE     ;enable a/d
0026 0008           00074         return
                    00075 ;
                    00076 ;This routine is a software delay of 10uS for the a/d setup.
                    00077 ;At 4Mhz clock, the loop takes 3uS, so initialize TEMP with
                    00078 ;a value of 3 to give 9uS, plus the move should result in
                    00079 ;a total time of > 10uS.
0027                00080 SetupDelay
0027 3003           00081         movlw   .3
0028 0090           00082         movwf   TEMP
0029                00083 SD
0029 0B90           00084         decfsz  TEMP, F
002A 2829           00085         goto    SD
002B 0008           00086         return
                    00087
                    00088 ;
                    00089
                    00090         END


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---X----------- XXXXXXXXXXXXXXXX XXXXXXXXXXXX---- ----------------

All other memory blocks unused.

Program Memory Words Used:    30
Program Memory Words Free:   994


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     2 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C

```
MPASM 01.40 Released            INTAD.ASM   1-16-1997  15:21:10         PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001
                     00002 ;TITLE   "Single channel A/D with interrupts"
                     00003 ;This program is a simple implementation of the PIC16C71's
                     00004 ;A/D. 1 Channel is selected (CH0). A/D interrupt is turned on,
                     00005 ;hence on completion of a/d conversion, an interrupt is generated.
                     00006 ;The A/D is configured as follows:
                     00007 ;       Vref = +5V internal.
                     00008 ;       A/D Osc. = internal RC Osc.
                     00009 ;       A/D Interrupt = On
                     00010 ;       A/D Channel = CH0
                     00011 ;
                     00012 ;The A/D result is displayed as a 8 bit value on 8 leds connected
                     00013 ;to port b. Hardware setup is the PICDEMO board.
                     00014 ;
                     00015 ;
                     00016 ;       Program:          INTAD.ASM
                     00017 ;       Revision Date:
                     00018 ;                         1-14-97      Compatibility with MPASMWIN 1.40
                     00019 ;
                     00020 ;
                     00021         LIST P=16C71
                     00022         ERRORLEVEL  -302
                     00023 ;
                     00024         include "p16c71.inc"
                     00001         LIST
                     00002 ; P16C71.INC  Standard Header File, Version 1.00  Microchip Technology
                     00142         LIST
                     00025 ;
  00000010           00026 flag    equ     10
  00000011           00027 TEMP    equ     11
  00000000           00028 adover  equ     0
  00000001           00029 adif    equ     1
  00000002           00030 adgo    equ     2
  00000006           00031 adie    equ     6
  00000007           00032 gie     equ     7
  00000005           00033 rp0     equ     5
                     00034 ;
0000                 00035         ORG     0x00
                     00036 ;
                     00037 ;
0000 2810            00038         goto    start
                     00039 ;
0004                 00040         org     0x04
0004 281E            00041         goto    service_ad      ;interrupt vector
                     00042 ;
                     00043 ;
0010                 00044         org     0x10
0010                 00045 start
0010 3000            00046         movlw   B'00000000'     ;init i/o ports
0011 0086            00047         movwf   PORTB
                     00048 ;       tris    PORTB
0012 1683            00049         BSF     STATUS, RP0     ; Bank1
0013 0086            00050         MOVWF   TRISB           ; PortB as outputs
0014 1283            00051         BCF     STATUS, RP0     ; Bank0
                     00052 ;
```

```
0015 2024           00053         call    InitializeAD
0016                00054 update
0016 1010           00055         bcf     flag,adover     ;reset software a/d flag
0017 202D           00056         call    SetupDelay      ;setup delay >= 10uS.
0018 1088           00057         bcf     ADCON0,adif     ;reset a/d int flag (ADIF)
0019 1508           00058         bsf     ADCON0,adgo     ;start new a/d conversion
001A 178B           00059         bsf     INTCON,gie      ;enable global interrupt
001B                00060 loop
001B 1810           00061         btfsc   flag,adover     ;a/d over?
001C 2816           00062         goto    update          ;yes start new conv.
001D 281B           00063         goto    loop            ;no then keep checking
                    00064 ;
001E                00065 service_ad
001E 1C88           00066         btfss   ADCON0,adif     ;ad interrupt?
001F 0009           00067         retfie                  ;no then ignore
0020 0809           00068         movf    ADRES,W         ;get a/d value
0021 0086           00069         movwf   PORTB           ;output to port b
0022 1410           00070         bsf     flag,adover     ;a/d done set
0023 0008           00071         return                  ;do not enable int
                    00072 ;
                    00073 ;
                    00074 ;InitializeAD, initializes and sets up the A/D hardware.
                    00075 ;select ch0 to ch3, RC OSC., a/d interrupt.
0024                00076 InitializeAD
0024 1683           00077         bsf     STATUS,rp0      ;select Bank1
0025 3000           00078         movlw   B'00000000'     ;select ch0-ch3...
0026 0088           00079         movwf   ADCON1          ;as analog inputs
0027 1283           00080         bcf     STATUS,rp0      ;select Bank0
0028 018B           00081         clrf    INTCON          ;clr all interrupts
0029 170B           00082         bsf     INTCON,adie     ;enable a/d int.
002A 30C1           00083         movlw   B'11000001'     ;select:RC osc,ch0...
002B 0088           00084         movwf   ADCON0          ;turn on a/d
002C 0008           00085         return
                    00086 ;
                    00087 ;This routine is a software delay of 10uS for the a/d setup.
                    00088 ;At 4Mhz clock, the loop takes 3uS, so initialize TEMP with
                    00089 ;a value of 3 to give 9uS, plus the move should result in
                    00090 ;a total time of > 10uS.
002D                00091 SetupDelay
002D 3003           00092         movlw   .3
002E 0091           00093         movwf   TEMP
002F                00094 SD
002F 0B91           00095         decfsz  TEMP, F
0030 282F           00096         goto    SD
0031 0008           00097         return
                    00098 ;
                    00099 ;
                    00100         END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---X---------- XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XX--------------

All other memory blocks unused.

Program Memory Words Used:    36
Program Memory Words Free:   988


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     2 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX D

```
MPASM 01.40 Released           MULTAD.ASM   1-16-1997  15:21:41         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;TITLE   "A/D using Multiple Channels"
                     00002 ;This program is a simple implementation of the PIC16C71's
                     00003 ;A/D feature. This program demonstrates
                     00004 ;how to select multiple channels on the PIC16C71.
                     00005 ;The A/D is configured as follows:
                     00006 ;       Vref = +5V internal.
                     00007 ;       A/D Osc. = internal RC osc.
                     00008 ;       A/D Interrupt = Off
                     00009 ;       A/D Channels = all in a "Round Robin" format.
                     00010 ;       A/D reuslts are stored in ram locations as follows:
                     00011 ;       ch0 --> ADTABLE + 0
                     00012 ;       ch1 --> ADTABLE + 1
                     00013 ;       ch2 --> ADTABLE + 2
                     00014 ;       ch3 --> ADTABLE + 3
                     00015 ;
                     00016 ;The ch0 A/D result is displayed as a 8 bit value on 8 leds
                     00017 ;connected to port b.
                     00018 ;Hardware: PICDEMO board.
                     00019 ;               Stan D'Souza 7/6/93.
                     00020 ;
                     00021 ;       Program:          MULTAD.ASM
                     00022 ;       Revision Date:
                     00023 ;                         1-14-97      Compatibility with MPASMWIN 1.40
                     00024 ;
                     00025 ;
                     00026         LIST P=16C71
                     00027         ERRORLEVEL  -302
                     00028 ;
                     00029         include "p16c71.inc"
                     00001         LIST
                     00002 ;P16C71.INC  Standard Header File, Version 1.00 Microchip Technology
                     00142         LIST
                     00030 ;
  00000010           00031 TEMP    EQU     10h
  00000001           00032 adif    equ     1
  00000002           00033 adgo    equ     2
                     00034 ;
  00000006           00035 ch2     equ     6
  00000007           00036 ch3     equ     7
  0000000C           00037 flag    equ     0C
  00000020           00038 ADTABLE equ     20
                     00039 ;
0000                 00040         ORG     0x00
                     00041 ;
                     00042 ;
0000 2810            00043         goto    start
                     00044 ;
0004                 00045         org     0x04
0004 2825            00046         goto    service_int      ;interrupt vector
                     00047 ;
                     00048 ;
0010                 00049         org     0x10
0010                 00050 start
0010 3000            00051         movlw   B'00000000'      ;make port b
```

```
0011 0086          00052         movwf   PORTB           ;as all outputs
                   00053 ;       tris    PORTB           ;       /
0012 1683          00054         BSF     STATUS, RP0      ; Bank1
0013 0086          00055         MOVWF   TRISB            ; PortB as outputs
0014 1283          00056         BCF     STATUS, RP0      ; Bank0
                   00057 ;
0015 2026          00058         call    InitializeAD
0016               00059 update
0016 0809          00060         movf    ADRES,W
0017 0080          00061         movwf   0                ;save in table
0018 3020          00062         movlw   ADTABLE          ;chk if ch0
0019 0204          00063         subwf   FSR,W            ;     /
001A 1D03          00064         btfss   STATUS,Z         ;yes then skip
001B 281E          00065         goto    NextAd           ;else do next channel
001C 0809          00066         movf    ADRES,W          ;get a/d value
001D 0086          00067         movwf   PORTB           ;output to port b
001E               00068 NextAd
001E 2030          00069         call    NextChannel      ;select next channel
001F 203C          00070         call    SetupDelay       ;set up > = 10uS
0020 1088          00071         bcf     ADCON0,adif      ;clear flag
0021 1508          00072         bsf     ADCON0,adgo      ;start new a/d conversion
0022               00073 loop
0022 1888          00074         btfsc   ADCON0,adif      ;a/d done?
0023 2816          00075         goto    update           ;yes then update
0024 2822          00076         goto    loop             ;wait till done
                   00077 ;
0025               00078 service_int
0025 0008          00079         return                   ;do not enable int
                   00080 ;
                   00081 ;
                   00082 ;InitializeAD, initializes and sets up the A/D hardware.
0026               00083 InitializeAD
0026 1683          00084         bsf     STATUS,5         ;select pg1
0027 3000          00085         movlw   B'00000000'      ;select ch0-ch3...
0028 0088          00086         movwf   ADCON1           ;as analog inputs
0029 1283          00087         bcf     STATUS,5         ;select pg0
002A 30C1          00088         movlw   B'11000001'      ;select:fosc/2, ch0.
002B 0088          00089         movwf   ADCON0           ;turn on a/d
002C 3020          00090         movlw   ADTABLE          ;get top of table address
002D 0084          00091         movwf   FSR              ;load into indirect reg
002E 0189          00092         clrf    ADRES            ;clr result reg.
002F 0008          00093         return
                   00094 ;
                   00095 ;NextChannel, selects the next channel to be sampled in a
                   00096 ;"round-robin" format.
0030               00097 NextChannel
0030 3008          00098         movlw   0x08             ;get channel offset
0031 0788          00099         addwf   ADCON0, F        ;add to conf. reg.
0032 1288          00100         bcf     ADCON0,5         ;clear any carry over
                   00101 ;increment pointer to correct a/d result register
0033 0190          00102         clrf    TEMP
0034 1988          00103         btfsc   ADCON0,3         ;test lsb of chnl select
0035 1410          00104         bsf     TEMP,0           ;set if ch1 or ch3
0036 1A08          00105         btfsc   ADCON0,4         ;test msb of chnl select
0037 1490          00106         bsf     TEMP,1           ;set if ch0 or ch2
0038 3020          00107         movlw   ADTABLE          ;get top of table
0039 0710          00108         addwf   TEMP,W           ;add with temp
003A 0084          00109         movwf   FSR              ;allocate new address
003B 0008          00110         return
                   00111 ;
                   00112 ;This routine is a software delay of 10uS for the a/d setup.
                   00113 ;At 4Mhz clock, the loop takes 3uS, so initialize TEMp with
                   00114 ;a value of 3 to give 9uS, plus the move etc should result in
                   00115 ;a total time of > 10uS.
003C               00116 SetupDelay
003C 3003          00117         movlw   .3
```


© 1997 Microchip Technology Inc.

```
003D 0090           00118         movwf   TEMP
003E                00119 SD
003E 0B90           00120         decfsz  TEMP, F
003F 283E           00121         goto    SD
0040 0008           00122         return
                    00123
                    00124 ;
                    00125
                    00126         END


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---X----------- XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : X--------------- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:    51
Program Memory Words Free:   973


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     2 suppressed
```

NOTES:

© 1997 Microchip Technology Inc.

Implementing Wake-up on Key Stroke

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

Microchip's PIC16CXXX microcontroller family are ideally suited to directly interface to a keypad. The high 4-bits of PORTB (RB7:RB4) have internal pull-ups and can trigger a "change on state" interrupt. This interrupt, if enabled, will wake the microcontroller from SLEEP. In most battery powered applications, a microcontroller is exercised when a key is pressed (e.g., in a remote keyless entry system). The life of the battery can be extended by using PIC16CXXX microcontrollers. This is done by putting the PIC16CXXX microcontroller into SLEEP mode for most of the time and wake-up only when a key is pressed.

IMPLEMENTATION

Figure 1 depicts an application where four keys are connected to RB7:RB4. Internal pull-ups are used to maintain a high level on these inputs. In this example, LEDs are connected to RB3:RB0. When SW1 is pressed, LED1 is turned on and when SW2 is pressed, LED2 is turned on and so on. The PIC16CXXX is normally in SLEEP mode with the "change on state" interrupt enabled. When SW1 is pressed, RB4 goes low and triggers an interrupt. Since the PIC16CXXX is in SLEEP, it first wakes up and starts executing code at the interrupt vector. Note that if the global interrupt is enabled, the program execution after an interrupt is at the interrupt vector, if the global interrupt is not enabled, the program starts executing the first line of code right after the `SLEEP` instruction.

After waking up, a 20 - 40 ms de-bounce delay is executed which checks the port for a key hit and, depending on which key is hit, its associated LED is turned on. The LEDs are used purely for demonstration purposes. In a remote keyless entry application, the remote code would be transmitted when the appropriate key is hit.

Figure 2 depicts a 4x4 keypad interface to a PIC16CXXX microcontroller. When using the PIC16CXXX in a keypad application, the internal pull-ups on RB7:RB4 can be enabled, eliminating the need for external pull-up resistors. The series 100Ω resistors are used for Electrostatic Discharge (ESD) protection, and are recommended in keypad interface applications.

SUMMARY

The PIC16CXXX is ideally suited to interface directly to a keypad application. Built in pull-up resistors and very low current consumption during sleep make it a very good candidate for battery powered remote operations and applications. Appendix A provides an example of the code.

Performance:

Code Size - 64 words

RAM Used - 0 bytes

FIGURE 1: 4 KEY INTERFACE TO PIC16CXXX

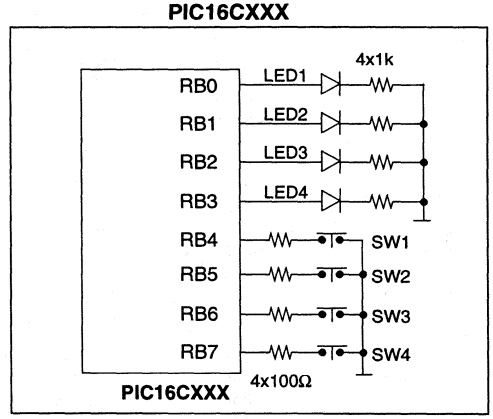
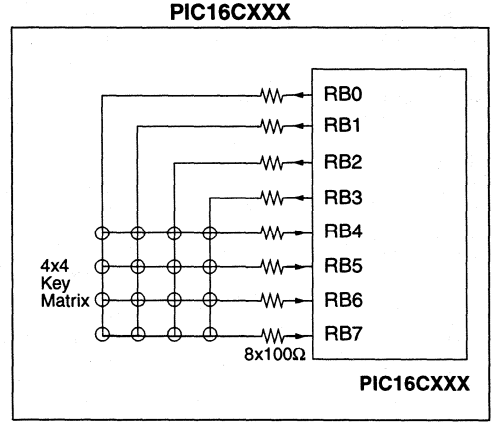


FIGURE 2: 4x4 KEYPAD INTERFACE TO PIC16CXXX



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A :

```
MPASM 01.40 Released            WAKUP.ASM   1-16-1997  16:04:19         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;This program demonstrates the wake-up on Keystroke feature of
                     00002 ;the PIC16C71. Port B pins RB4 - RB7 can be configured as inputs with
                     00003 ;internal pull up resistors, also the interrupt associated with the
                     00004 ;change on input on RB4 - RB7 can be set up to wake the chip from
                     00005 ;sleep.  If the global interrupt is enabled just before sleep, the
                     00006 ;program will vector to the interrupt vector (0004). If not, the chip
                     00007 ;will continue execution just after the next instruction following
                     00008 ;sleep. In this example code, port B is initialized to input 4
                     00009 ;push-buttons at RB4 - RB7. RB0 - RB3 are configured to drive LEDs
                     00010 ;corresponding to which pushbutton is hit (LED on RB0 when RB4 is
                     00011 ;hit and so on). Sleep is executed. When any key is hit, the
                     00012 ;processor wakes up and jumps to the interrupt vector. The
                     00013 ;corresponding LED is turned on and after the key is released, the
                     00014 ;whole process is repeated.
                     00015 ;
                     00016 ;       Program:          WAKEUP.ASM
                     00017 ;       Revision Date:
                     00018 ;                         1-16-97      Compatibility with MPASMWIN 1.40
                     00019 ;
                     00020 ;*******************************************************************
                     00021 ;
                     00022         LIST P=16C71
                     00023 ;
  00000002           00024 z       equ     2
  00000007           00025 RBPU    equ     7
  00000010           00026 temp    equ     10h
  00000001           00027 OptionReg  equ     1h
  00000001           00028 F       EQU     1
                     00029 ;
                     00030         include "p16c71.inc"
                     00001         LIST
                     00002 ;P16C71.INC Standard Header File, Version 1.00 Microchip Technology
                     00142         LIST
                     00031 ;
0000                 00032         org     0
0000 2805            00033         goto    start
                     00034 ;
0004                 00035         org     4
0004 2808            00036         goto    ServiceInterrupt
                     00037 ;
                     00038 ;
0005                 00039 start
0005 2024            00040         call    InitPortB       ;initialize port B
0006                 00041 loop
                     00042         sleep                   ;sleep till key is hit
0006 0000            00043         nop
0007 2806            00044         goto    loop
                     00045 ;
0008                 00046 ServiceInterrupt
0008 180B            00047         btfsc   INTCON,RBIF     ;change on rb int?
0009 280D            00048         goto    ServiceWakup    ;yes then service
000A 128B            00049         bcf     INTCON,T0IE     ;clear TMR0 int mask
000B 110B            00050         bcf     INTCON,T0IF     ;clear flag
000C 0008            00051         return
```


© 1997 Microchip Technology Inc.

```
                    00052 ;
                    00053 ;This routine checks which key is hit and lights up the
                    00054 ;corresponding LED associated with it. eg. RB0's LED when
                    00055 ;RB4's key is pressed. Finally it waits till all keys have
                    00056 ;been released before returning form the service routine.
000D                00057 ServiceWakup
000D 118B           00058         bcf     INTCON,RBIE     ;clear mask
000E 0906           00059         comf    PORTB,W         ;read PORTB
000F 100B           00060         bcf     INTCON,RBIF     ;clear flag
0010 2035           00061         call    delay16         ;do de-bounce for 16mSecs
0011 0906           00062         comf    PORTB,W         ;read port B again
0012 39F0           00063         andlw   B'11110000'     ;mask outputs
0013 0090           00064         movwf   temp            ;save in temp
0014 0E10           00065         swapf   temp,W          ;switch low and high
0015 0086           00066         movwf   PORTB           ;send as outputs.
0016 2018           00067         call    KeyRelease      ;check for key release
0017 0009           00068         retfie
                    00069 ;
                    00070 ;This sub-routine, waits till all key have been released
                    00071 ;In order to save power, the chip is in sleep mode till
                    00072 ;all keys are released.
0018                00073 KeyRelease
0018 2035           00074         call    delay16         ;do debounce
0019 0906           00075         comf    PORTB,W         ;read PORTB
001A 100B           00076         bcf     INTCON,RBIF     ;clear flag
001B 158B           00077         bsf     INTCON,RBIE     ;enable mask
001C 39F0           00078         andlw   B'11110000'     ;clear outputs
001D 1903           00079         btfsc   STATUS,z        ;key still pressed?
001E 0008           00080         return                  ;no then return
001F 0063           00081         sleep                   ;else save power
0020 118B           00082         bcf     INTCON,RBIE     ;on wake up clear mask
0021 0906           00083         comf    PORTB,W
0022 100B           00084         bcf     INTCON,RBIF     ;clear flag
0023 2818           00085         goto    KeyRelease      ;try again
                    00086 ;
                    00087 ;
                    00088 ;This sub-routine, initializes PortB.
0024                00089 InitPortB
0024 1683           00090         bsf     STATUS,RP0      ;select bank1
0025 3003           00091         movlw   B'00000011'     ;Port_A digital I/O
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0026 0088           00092         movwf   ADCON1          ;               /
0027 3000           00093         movlw   0               ;
0028 0085           00094         movwf   PORTA           ;set port a as outputs
0029 30F0           00095         movlw   B'11110000'     ;RB0-RB3 outputs
002A 0086           00096         movwf   PORTB           ;RB4-RB7 inputs
002B 1381           00097         bcf     OptionReg,RBPU  ;enable pull up
002C 1283           00098         bcf     STATUS,RP0      ;select page 0
002D 0186           00099         clrf    PORTB           ;init port B
002E 0185           00100         clrf    PORTA           ;make port a all low
002F 1405           00101         bsf     PORTA,0         ;make first bit high
0030 118B           00102         bcf     INTCON,RBIE     ;disable mask
0031 0806           00103         movf    PORTB,W         ;read port
0032 100B           00104         bcf     INTCON,RBIF     ;clear flag
0033 158B           00105         bsf     INTCON,RBIE     ;enable mask
0034 0009           00106         retfie                  ;enable global and return
                    00107 ;
                    00108 ;delay16 waits for approx 16.4mSecs using TMR0 interrupts
                    00109 ;fosc speed is 4Mhz.
0035                00110 delay16
0035 1683           00111         bsf     STATUS,RP0      ;select Bank1
0036 3007           00112         movlw   B'00000111'     ;fosc/256 --> TMR0
0037 0081           00113         movwf   OptionReg       ;       /
0038 1283           00114         bcf     STATUS,RP0      ;select Bank0
0039 0181           00115         clrf    TMR0
003A 110B           00116         bcf     INTCON,T0IF     ;clear flag
```

```
003B 168B           00117         bsf     INTCON,T0IE     ;enable mask
003C                00118 CheckAgain
003C 1D0B           00119         btfss   INTCON,T0IF     ;timer overflowed?
003D 283C           00120         goto    CheckAgain      ;no check again
003E 128B           00121         bcf     INTCON,T0IE     ;else clear mask
003F 110B           00122         bcf     INTCON,T0IF     ;clear flag
0040 0008           00123         return
                    00124 ;
                    00125         end
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : X-------------- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:    62
Program Memory Words Free:   962


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     1 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN554

Software Implementation of I²C™ Bus Master

Author: *Amar Palacherla*
Microchip Technology Inc.

INTRODUCTION

This application note describes the software implementation of I²C interface routines for the PIC16CXXX family of devices. Only the master mode of I²C interface is implemented in this application note. This implementation is for a single master communication to multiple slave I²C devices.

Some PIC16CXXX devices, such as the PIC16C64 and PIC16C74, have on-chip hardware which implements the I²C slave interface, while other PIC16CXXX devices, such as the PIC16C71 and PIC16C84, do not have the same on-chip hardware.

This application note does not describe the I²C Bus specifications and the user is assumed to have an understanding of the I²C Bus. For detailed information on the bus, the user is advised to read the I²C Bus Specification document from Philips/Signetics (order number 98-8080-575). The I²C Bus is a two-wire serial bus with multiple possible masters and multiple possible slaves connected to each other through two wires. The two wires consists of a clock line (SCL) and a data line (SDA) with both lines being bi-directional. Bi-directional communication is facilitated through the use of wire and connection (the lines are either active-low or passive high). The I²C Bus protocol also allows collision detection, clock synchronization and hand-shaking for multi-master systems. The clock is always generated by the master, but the slave may hold it low to generate a wait state.

In most systems the microcontroller is the master and the external peripheral devices are slaves. In these cases this application note can be used to attach I²C slaves to the PIC16CXXX (the master) microcontroller. The multi-master system is not implemented because it is extremely difficult to meet all the I²C Bus timing specifications using software. For a true slave or multi-master system, some interface hardware is necessary (like START & STOP bit detection).

In addition to the low level single master I²C routines, a collection of high level routines with various message structures is given. These high level macros/routines can be used as canned routines to interface to most I²C slave devices. As an example, the test program talks to two Serial EEPROMs (Microchip's 24LC04 and 24LC01).

IMPLEMENTATION

Two levels of software routines are provided. The low-level routines "`i2c_low.asm`" are provided in Appendix A and the high level routines "`i2c_high.asm`" are provided in Appendix B.

The messages passed (communicated on the two wire network) are abbreviated and certain notation is used to represent Start, Stop and other conditions. These abbreviations are described in Table 1.

TABLE 1: DESCRIPTION OF ABBREVIATIONS USED

Abbreviations	Explanation
S	Start Condition
P	Stop Condition
SlvAR	Slave Address (for read operation)
SlvAW	Slave Address (for write operation)
A	Acknowledge condition (positive ACK)
N	Negative Acknowledge condition (NACK)
D	Data byte, D[0] represents byte 0, D[1] represents second byte

© 1997 Microchip Technology Inc.

Message Format

In the high level routines, the basic structure of the message is given. Every I²C slave supports one or more message structures. For example, Microchip's 24LC04 Serial EEPROM supports the following message (to write a byte to Serial EEPROM at current address counter) S-SlvAW-A-D-A-P which basically means the following sequence of operations are required:

a) Send Start Bit
b) Send Slave Address for Write Operations
c) Expect Acknowledge
d) Send Data Byte
e) Expect Acknowledge
f) Issue a STOP Condition

Slave Address

Both 10-bit and 7-Bit addressing schemes are implemented as specified by the I²C Bus specification. Before calling a certain sub-routine (high level or low-level), the address of the slave being addressed must be loaded using either "LOAD_ADDR_8" (for 7-bit address slaves) or "LOAD_ADDR_10" macro (for 10-bit address slaves). These macros not only load the address of the slaves for all the following operations, but also setup conditions for 7- or 10-bit addressing modes. See the macros section for more details.

CLOCK STRETCHING

In the I²C Bus, the clock (SCL line) is always provided by the master. However, the slave can hold the line low even though the master has released it. The master must check this condition and wait for the slave to release the clock line. This provides a built-in wait state for the I²C Bus. This feature is implemented and can be turned on or off as an assembly time option (by configuring the _ENABLE_BUS_FREE_TIME flag to be TRUE or FALSE). If the clock is held low for too long, say 1 ms, then an error condition is assumed and a T0CKI interrupt is generated.

ARBITRATION

The I²C Bus specifies both bit-by-bit and byte mode arbitration procedures for multi-master systems. However, the arbitration is not needed in a single master system, and therefore is not implemented in this application note.

HARDWARE

Two I/O pins are used to emulate the Clock Line, SCL, and the Data Line, SDA. In the example test program, RB0 is used as the SCL line and RB1 as the SDA line. On initialization, these I/O lines are configured as input pins (tri-state) and their respective latches are loaded with '0's. To emulate the high state (passive), these lines are configured as inputs. To emulate the active low state, the pins are configured as outputs (with the assumption of having external pull-up resistors on both lines).

For devices that have the on-chip I²C hardware (SSP module), slope control of the I/O is implemented on the SCK and SDA pins. For software not implemented on the SCK and SDA pins of the SSP module, external components for slope control of the I/O may be required by the system.

© 1997 Microchip Technology Inc.

I²C ROUTINES

Status Register (File Register "Bus_Status")

The bit definitions of the status register are described in the table given below. These bits reflect the status of the I²C Bus.

Bit #	Name	Description
0	_Bus_Busy	1 = Start Bit transmitted 0 = STOP condition
1	_Abort	It is set when a fatal error condition is detected. The user must clear this bit. This bit is set when the clock line, SCL, is stuck low.
2	_Txmt_Progress	1 = transmission in progress
3	_Rcv_Progress	1 = reception in progress
4	_Txmt_Success	1 = transmission successfully completed 0 = error condition
5	_Rcv_Success	1 = reception successfully completed 0 = error condition
6	_Fatal_Error	1 = FATAL error occurred (the communication was aborted).
7	_$\overline{\text{ACK}}$_Error	1 = slave sent $\overline{\text{ACK}}$ while the master was expecting an ACK. This may happen for example if the slave was not responding to a message.

Control Register (File Register "Bus_Control")

The bit definitions of the control register are described in the table given below. These bits must be set by the software prior to performing certain operations. Some of the high level routines described later in this section set these bits automatically.

Bit #	Name	Description
0	_10BitAddr	1 = 10-bit slave addressing 0 = 7-bit addressing.
1	_Slave_RW	1 = READ operation 0 = WRITE operation.
2	_Last_Byte_Rcv	1 = last byte must be received. Used to send $\overline{\text{ACK}}$.
3, 4, 5	—	Unused bits, can be used as general purpose bits.
6	_SlaveActive	A status bit indicating if a slave is responding. This bit is set or cleared by calling the I2C_TEST_DEVICE macro. See description of this I2C_TEST_DEVICE macro.
7	_TIME_OUT_	A status bit indicating if a clock is stretched low for more than 1 ms, indicating a bus error. On this time out, the operation is aborted.

Lower Level Routines

Function Name	Description
InitI2CBus_Master	Initializes Control/Status Registers, and set SDA & SCL lines. Must be called on initialization.
TxmtStartBit	Transmits a START (S) condition.
TxmtStopBit	Transmits a STOP (P) condition.
LOAD_ADDR_8	The 7-bit slave's address must be passed as a constant parameter.
LOAD_ADDR_10	The 10-bit slave's address must be passed as a constant parameter.
Txmt_Slave_Addr	Transmits a slave address. Prior to calling this routine, the address of the slave being addressed must be loaded using LOAD_ADDR_8 or LOAD_ADDR_10 routines. Also the Read/Write condition must be set in the control register.
SendData	Transmits a byte of data. Prior to calling this routine, the byte to be transmitted must be loaded into DataByte file register.
GetData	Receives a byte of data in DataByte file register. If the data byte to be received is the last byte, the _Last_Byte_Rcv bit in control register must be set prior to calling this routine.

© 1997 Microchip Technology Inc.

MACROS

High Level

The high level routines are implemented as a mixture of function calls and macros. These high level routines call the low level routines described previously. In most cases only a few of the high level routines may be needed and the user can remove or not include the routines that are not necessary to conserve program memory space. Examples are given for a few functions.

I2C_TEST_DEVICE

Parameters : None

Purpose : To test if a slave is present on the network.

Description : Before using this macro, the address of the slave being tested must be loaded using LOAD_ADDR_8 or LOAD_ADDR_10 macro. If the slave under test is present, then "_SlaveActive" status bit (in Bus_Control file register) is set. If not, then this bit is cleared, indicating that the slave is either not present on the network or is not listening.

Message : S-SlvAW-A-P

Example :

```
LOAD_ADDR_8       0xA0                  ; 24LC04 address
I2C_TEST_DEVICE
btfss             _SlaveActive          ; See If slave is responding
goto              SlaveNotPresent       ; 24LC04 is not present
                                        ; Slave is present
                                        ; Continue with program
```

I2C_WR

Parameters : _BYTES_, _SourcePointer_

BYTES Number of bytes starting from RAM pointer _SourcePointer_

SourcePointer Data Start Buffer pointer in RAM (file registers)

Purpose : A basic macro for writing a block of data to a slave

Description : This macro writes a block of data (# of bytes = _BYTES_) to a slave. The starting address of the block of data is _SourcePointer_. If an error occurs, the message is aborted and the user must check Status flags (e.g. _Txmt_Success bit)

Message : S-SlvAW-A-D[0]-A.....A-D[N-1]-A-P

Example :

```
btfsc             _Bus_Busy             ; Check if bus is free
goto              $-1
LOAD_ADDR_8       _Slave_1_Addr
I2C_WR            0x09, DataBegin       ;
```


© 1997 Microchip Technology Inc.

I2C_WR_SUB

Parameters : _BYTES_, _SourcePointer_, _Sub_Address_
- _BYTES_ Number of bytes starting from RAM pointer _SourcePointer_
- _SourcePointer_ Data Start Buffer pointer in RAM (file Registers)
- _Sub_Address_ Sub-address of the Slave

Purpose : Write a block of data to a slave starting at slave's sub-addr

Description : Same as `I2C_WR` function, except that the starting address of the slave is also specified. For example, while writing to an I^2C Memory Device, the sub-addr specifies the starting address of the memory. The I^2C may prove to be more efficient than this macro in most situations. Advantages will be found for Random Address Block Writes for Slaves with Auto Increment Sub-addresses (like Microchip's 24CXX series Serial EEPROMs).

Message: S-SlvAW-A-SubA-A-D[0]-A.....A-D[N-1]-A-P

Example :

```
LOAD_ADDR_8   _Slave_2_Addr                ; Load addr of 7-bit slave
I2C_WR_SUB    0x08, DataBegin+1, 0x30
```

In the above example, 8 Bytes of data starting from addr (DataBegin+1) is written to 24LC04 Serial EEPROM beginning at 0x30 address

I2C_WR_SUB_SWINC

Parameters : _BYTES_, _SourcePointer_, _Sub_Address_
- _BYTES_ Number of bytes starting from RAM pointer _SourcePointer_
- _SourcePointer_ Data Start Buffer pointer in RAM (file Registers)
- _Sub_Address_ Sub-address of the Slave

Purpose : Write a block of data to a slave starting at slave's sub-addr

Description : Same as `I2C_WR_SUB` function, except that the sub-address (incremented) is sent after every data byte. A very inefficient message structure and the bus is given up after each data byte. This is useful for when the slave does not have an auto-increment sub-address feature.

Message : *S-SlvAW-A-(SubA+0)-A-D[0]-A-P*
S-SlvAW-A-(SubA+1)-A-D[1]-A-P
and so on until #of Bytes

© 1997 Microchip Technology Inc.

I2C_WR_BYTE_MEM

Parameters : _BYTES_, _SourcePointer_, _Sub_Address_

BYTES	Number of bytes starting from RAM pointer _SourcePointer_
SourcePointer	Data Start Buffer pointer in RAM (file Registers)
_Sub_Address_	Sub-address of the Slave

Purpose : Write a block of data to a slave starting at slave's sub-address

Description : Same as `I2C_WR_SUB_SWINC`, except that a delay is added between each message. This is necessary for some devices, like EEPROMs, which accept only a byte at a time for programming (devices without on-chip RAM buffer) and after each byte a delay is necessary before a next byte is written.

Message :

S-SlvAW-A-(SubA+0)-A-D[0]-A-P
Delay 1 ms
S-SlvAW-A-(SubA+1)-A-D[1]-A-P
Delay 1 ms
•
•
•
and so on until #of Bytes

I2C_WR_BUF_MEM

Parameters : _BYTES_, _SourcePointer_, _Sub_Address_, _Device_BUF_SIZE_

BYTES	Number of bytes starting from RAM pointer _SourcePointer_
SourcePointer	Data Start Buffer pointer in RAM (file Registers)
_Sub_Address_	Sub-address of the Slave
_Device_BUF_SIZE_	the slaves on-chip buffer size

Purpose : Write a block of data to a slave starting at slave's sub-addr

Description : This Macro/Function writes #of _BYTES_ to an I^2C memory device. However some devices, especially EEPROMs, must wait while the device enters into programming mode. But some devices have an on-chip temperature data hold buffer and is used to store data before the device actually enters into programming mode. For example, the 24C04 series of Serial EEPROMs from Microchip have an 8-byte data buffer. So one can send 8 bytes of data at a time and then the device enters programming mode. The master can either wait until a fixed time and then retry to program or can continuously poll for $\overline{ACK}$ bit and then transmit the next block of data for programming.

Message : `I2C_SUB_WR` operations are performed in loop and each time data buffer of BUF_SIZE is output to the device. Then the device is checked for busy and when not busy another block of data is written.

© 1997 Microchip Technology Inc.

I2C_READ

Parameters : _BYTES_, _DestPointer_

BYTES Number of bytes starting from RAM pointer _SourcePointer_

DestPointer Data Start Buffer pointer in RAM (file Registers)

Purpose : A basic macro for reading a block of data from a slave

Description : This macro reads a block of data (number of bytes = _BYTES_) from a slave. The starting address of the block of data is _DestPointer_. If an error occurs, the message is aborted and the user must check Status flags (e.g. _Rcv_Success bit). Note that on the last byte to receive, NACK is sent.

Message : S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P

Example :

```
LOAD_ADDR_10    _Slave_3_Addr
I2C_READ_SUB    8, DataBegin
btfss           _Rcv_Success
goto            ReceiveError
goto            ReceiveSuccess
```

In the example above, 8 bytes of data is read from a 10-bit slave and stored in the master's RAM starting at address DataBegin.

© 1997 Microchip Technology Inc.

I2C_READ_SUB

Parameters : _BYTES_, _DestPointer_, _SubAddress

BYTES Number of bytes starting from RAM pointer _SourcePointer_
DestPointer Data Start Buffer pointer in RAM (file Registers)
SubAddress Sub-address of the slave

Purpose : A basic macro for reading a block of data from a slave

Description : This macro reads a block of data (# of bytes = _BYTES_) from a slave starting at slave's sub-address _SubAddress. The data received is stored in master's RAM starting at address _DestAddress. If an error occurs, the message is aborted and the user must check Status flags (e.g. _Rcv_Success bit).

This MACRO/Subroutine reads a message from a slave device preceded by a write of the sub-address between the sub-address write and the following reads, a STOP condition is not issued and a "REPEATED START" condition is used so that another master will not take over the bus, and also that no other master will overwrite the sub-address of the same slave. This function is very commonly used in accessing Random/Sequential reads from a memory device (e.g., 24CXX serial of Serial EEPROMs from Microchip).

Message : *S-SlvAW-A-SubAddr-A-S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P*

Example :

```
LOAD_ADDR_10  _Slave_3_Addr
I2C_READ      8,  DataBegin, 0x60
btfss         _Rcv_Success
goto          ReceiveError
goto          ReceiveSuccess
```

In the example above, 8 bytes of data is read from a 10-bit slave (starting at address 0x60h) and stored in the master's RAM starting at address DataBegin.

I2C_READ_BYTE *or* I2C_READ_STATUS

Parameters : _DestPointer_

DestPointer Data Start Buffer pointer in RAM (file Registers)

Purpose : To read a Status Byte from Slave

Description : Several I^2C Devices can send a Status Byte upon reception of the control byte. This Macro reads a Status byte from a slave to the master's RAM location _DestPointer_. This function is basically the same as `I2C_READ` for a single byte read. As an example of this command, the 24Cxx serial Serial EEPROM from Microchip will send the memory data at the current location when `I2C_READ_STATUS` function is called. On successful operation of this command, W register = '1' else W register = '0' on errors.

Message : *S-SlvAR-A-D-A-N-P*

© 1997 Microchip Technology Inc.

I2C_WR_SUB_WR

Parameters : _Bytes1_, _SrcPtr1_, _SubAddr_, _Bytes2_, _SrcPtr2_

Bytes1	Number of bytes in the first data block
SrcPtr1	Starting address of first data block
SubAddr	Sub-address of the slave
Bytes2	Number of bytes in the second data block
SrcPtr2	Starting address of second data block

Purpose : To send two blocks of data to a slave at its sub address

Description : This Macro writes two blocks of data (of variable length) starting at slave's sub-address _SubAddr_. This Macro is essentially the same as calling `I2C_WR_SUB` twice, but a STOP bit is not sent in-between the transmission of the two blocks. This way the Bus is not given up.

This function may be useful for devices which need two blocks of data in which the first block may be an extended address of a slave device. For example, a large I^2C memory device, or a teletext device with an extended addressing scheme, may need multiple bytes of data in the first block that represents the actual physical address and is followed by a second block that actually represents the data.

Message : *S-SlvW-A-SubA-A-D1[0]-A-....-D1[N-1]-A-D2[0]-A-.....A-D2[M-1]-A-P*

I2C_WR_SUB_RD

Parameters : _Count1_, _SrcPtr_, _SubAddr_, _Count2_, _DestPtr_

Count1	Length of the source buffer
SrcPtr	Source pointer address
SubAddr	Sub-address of the slave
Count2	Length of the destination buffer
DestPtr	Address of destination buffer

Purpose : To send a block of data and then receive a block of data

Description : This macro writes a block of data (of length _Count1_) to a slave starting at sub-address _SubAddr_ and then reads a block of data (of length _Count2_) to the master's destination buffer (starting at address _DestPtr_). Although this operation can be performed using previously defined Macros, this function does not give up the bus in between the block writes and block reads. This is achieved by using the Repeated Start Condition.

Message : *S-SlvW-A-SubA-A-D1[0]-A-.....-A-D1[N-1]-A-S-SlvR-A-D2[0]-A-.....A-D2[M-1]-N-P*

© 1997 Microchip Technology Inc.

I2C_WR_COM_WR

Parameters : _Count1_, _SrcPtr1_, _Count2_, _SrcPtr2_

Count1	Length of the first data block
SrcPtr1	Source pointer of the first data block
Count2	Length of the second data block
SrcPtr2	Source pointer of the second data block

Purpose : To send two blocks of data to a slave in one message.

Description : This macro writes a block of data (of length _Count1_) to a slave and then sends another block of data (of length _Count2_) without giving up the bus.

For example, this kind of transaction can be used in an I^2C LCD driver where a block of control and address information is needed and then another block of actual data to be displayed is needed.

Message : *S-SlvW-A-D1[0]-A-.....A-D1[N-1]-A-D2[0]-A-.....-A-D2[M-1]-A-P*

APPLICATIONS

The I^2C Bus is a simple two wire serial protocol for inter-IC communications. Many peripheral devices (acting as slaves) are available in the market with I^2C interface (e.g., serial EEPROM, clock/calendar, I/O Port expanders, LCD drivers, A/D converters). Although some of the PIC16CXXX devices do not have on-chip I^2C hardware interface, due to the high speed throughput of the microcontroller (250 ns @ 16 MHz input clock), the I^2C bus can be implemented using software.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: I²C.H

```
MPASM 01.40.01 Intermediate  I2C_TEST.ASM   4-4-1997  14:47:18         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         Title           "I2C Slave Mode Implemetation"
                     00002         SubTitle        "Rev 0.2        :  04 April 1997"
                     00003
                     00004
                     00005
                     00006 ;*****************************************************************
                     00007 ;
                     00008 ;                       I2C Slave Mode Using Software Polling
                     00009 ;
                     00010 ;  Start Bit Is detected by connecting SDA to RB0/INT Pin
                     00011 ;  Other bits, including STOP & Repeated Start Conditions are Software Polled
                     00012 ;
                     00013 ;    The software is implemented using PIC16C84 & thus can be ported to all
                     00014 ;       Enhanced core PIC16CXX products
                     00015 ;
                     00016 ;RB1 is SCL      (Any I/O Pin May Be used instead)
                     00017 ; RB0/INT is SDA (Must use this pin for START bit detect when in idle mode)
                     00018 ;
                     00019 ;   Since Slave Mode is always Application Dependent, as a Test Program, PIC16C84 is used to
                     00020 ;    emulate partial functionality of Microchip's 24Cxx Serial EEPROMs
                     00021 ;
                     00022 ;
                     00023 ;       Program:          I2C_TEST.ASM
                     00024 ;       Revision Date:    Rev 0.1       :  01 Mar 1993
                     00025 ;                         4-04-97       Compatibility with MPASMWIN 1.40
                     00026 ;
                     00027 ;*****************************************************************
                     00028 ;
                     00029 #define _MY_ADDRESS     0xD6    ; This slave's address
                     00030 ;
                     00031 #define AtoD_Slave      1
                     00032 #define EE_Slave        0
                     00033
                     00034 ;
```


© 1997 Microchip Technology Inc.

```
                00035
                00036                   LIST    p = 16C71
                00037                   ERRORLEVEL  - 302
                00038
                00039                   Radix   DEC
                00040                   EXPAND
                00041
00F42400        00042 _ClkIn            equ     16000000
003D0900        00043 _ClkOut           equ     (_ClkIn >> 2)
                00044 ;
                00045 ;
00000020        00046 ControlByte       EQU     0x20
                00047 #define          _STOP   ControlByte, 0
                00048 #define          _START  ControlByte, 1
                00049 #define          _RW     ControlByte, 2
                00050 #define          _ACK    ControlByte, 3
00000021        00051 SlvStatus         EQU     0x21
                00052 ;
                00053 ;
                00054
                00055                   include         "p16c71.inc"
                00001           LIST
                00002 ; P16C71.INC  Standard Header File, Version 1.00    Microchip Technology, Inc.
                00142           LIST
                00056
00000008        00057 _eedata set       0x08
00000009        00058 _eeadr  set       0x09
                00059
00000008        00060 _eecon1 set       0x08
00000009        00061 _eecon2 set       0x09            ; bank 1
                00062
                00063 #define          _rd     _eecon1,0
                00064 #define          _wr     _eecon1,1
                00065 #define          _wren   _eecon1,2
                00066 #define          _wrerr  _eecon1,3
                00067 #define          _eeif   _eecon1,4
                00068
00000000        00069 LSB      equ     0
00000007        00070 MSB      equ     7
                00071
                00072                   include "i2c.h"
                00001 ;*******************************************************************************************************
                      *
                00002 ;                                 I2C Bus Header File
                00003 ;*******************************************************************************************************
                      *
                00004
```

© 1997 Microchip Technology Inc.

```
003D0900          00005 _ClkOut         equ     (_ClkIn >> 2)
                  00006
                  00007 ;
                  00008 ; Compute the delay constants for setup & hold times
                  00009 ;
00000010          00010 _40uS_Delay     set     (_ClkOut/250000)
00000012          00011 _47uS_Delay     set     (_ClkOut/212766)
00000014          00012 _50uS_Delay     set     (_ClkOut/200000)
                  00013
                  00014 #define _OPTION_INIT    (0xC0 | 0x03)          ; Prescaler to TMR0 for Appox 1 mSec timeout
                  00015 ;
                  00016 #define _SCL    PORTB,0
                  00017 #define _SDA    PORTB,1
                  00018
                  00019 #define _SCL_TRIS       _trisb,0
                  00020 #define _SDA_TRIS       _trisb,1
                  00021
                  00022 #define _WRITE_         0
                  00023 #define _READ_          1
                  00024
                  00025
                  00026 ;                                       Register File Variables
                  00027
                  00028
                  00029                 CBLOCK  0x0C
0000000C          00030                         SlaveAddr               ; Slave Addr must be loaded into this reg
0000000D          00031                         SlaveAddrHi             ; for 10 bit addressing mode
0000000E          00032                         DataByte                ; load this reg with the data to be transmitted
0000000F          00033                         BitCount                ; The bit number (0:7) transmitted or received
00000010          00034                         Bus_Status              ; Status Reg of I2C Bus for both TXMT & RCVE
00000011          00035                         Bus_Control             ; control Register of I2C Bus
00000012          00036                         DelayCount
00000013          00037                         DataByteCopy            ; copy of DataByte for Left Shifts (destructive)
                  00038
00000014          00039                         SubAddr                 ; sub-address of slave (used in I2C_HIGH.ASM)
00000015          00040                         SrcPtr                  ; source pointer for data to be transmitted
                  00041
00000016          00042                         tempCount               ; a temp variable for scratch RAM
00000017          00043                         StoreTemp_1             ; a temp variable for scratch RAM, do not disturb contents
                  00044
00000018          00045                         _End_I2C_Ram            ; unused, only for ref of end of RAM allocation
                  00046                 ENDC
                  00047
                  00048 ;*******************************************************************************
                  00049 ;                       I2C Bus Status Reg Bit Definitions
                  00050 ;*******************************************************************************
                  00051
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
00052 #define _Bus_Busy       Bus_Status,0
00053 #define _Abort          Bus_Status,1
00054 #define _Txmt_Progress  Bus_Status,2
00055 #define _Rcv_Progress   Bus_Status,3
00056
00057 #define _Txmt_Success   Bus_Status,4
00058 #define _Rcv_Success    Bus_Status,5
00059 #define _Fatal_Error    Bus_Status,6
00060 #define _ACK_Error      Bus_Status,7
00061
00062 ;*****************************************************************************
00063 ;                       I2C Bus Control Register
00064 ;*****************************************************************************
00065 #define _10BitAddr      Bus_Control,0
00066 #define _Slave_RW       Bus_Control,1
00067 #define _Last_Byte_Rcv  Bus_Control,2
00068
00069 #define _SlaveActive    Bus_Control,6
00070 #define _TIME_OUT_      Bus_Control,7
00071
00072
00073
00074
00075 ;*****************************************************************************************************
     *
00076 ;                               General Purpose Macros
00077 ;*****************************************************************************************************
     *
00078
00079 RELEASE_BUS     MACRO
00080                         bsf     STATUS,RP0      ; select Bank1
00081                         bsf     _SDA            ; tristate SDA
00082                         bsf     _SCL            ; tristate SCL
00083 ;                       bcf     _Bus_Busy       ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
00084                 ENDM
00085
00086 ;*****************************************************************************************************
     *
00087 ;                       A MACRO To Load 8 OR 10 Bit Address To The Address Registers
00088 ;
00089 ;  SLAVE_ADDRESS is a constant and is loaded into the SlaveAddress Register(s) depending
00090 ;  on 8 or 10 bit addressing modes
00091 ;*****************************************************************************************************
     *
00092
00093 LOAD_ADDR_10    MACRO   SLAVE_ADDRESS
00094
```

```
                        00095                   bsf     _10BitAddr                                      ; Slave has 10 bit address
                        00096                   movlw   (SLAVE_ADDRESS & 0xff)
                        00097                   movwf   SlaveAddr                                       ; load low byte of address
                        00098                   movlw   ((SLAVE_ADDRESS >> 7) & 0x06) | 0xF0)           ; 10 bit addr is 11110XX0
                        00099                   movwf   SlaveAddr+1                                     ; hi order  address
                        00100
                        00101                   ENDM
                        00102
                        00103 LOAD_ADDR_8       MACRO   SLAVE_ADDRESS
                        00104
                        00105                   bcf     _10BitAddr      ; Set for 8 Bit Address Mode
                        00106                   movlw   (SLAVE_ADDRESS & 0xff)
                        00107                   movwf   SlaveAddr
                        00108
                        00109                   ENDM
                        00073
                        00074
                        00075                   CBLOCK  _End_I2C_Ram
  00000018              00076                           SaveStatus                                      ; copy Of STATUS Reg
  00000019              00077                           SaveW register                                  ; copy of W register
  0000001A              00078                           byteCount
  0000001B              00079                           HoldData
                        00080                   ENDC
                        00081
                        00082                   CBLOCK  0x20
  00000020              00083                           DataBegin                                       ; Data to be read or written is stored here
                        00084                   ENDC
                        00085
                        00086 ;*************************************************************************************
                        00087
0000                    00088                   ORG     0x00
                        00089
0000 2815               00090                   goto    Start
                        00091 ;
0004                    00092                   ORG     0x04
                        00093 ;*************************************************************************************************
                        00094 ;                               Interrupt Service Routine
                        00095 ;
                        00096 ;   For I2C Slave routines, only Rb0/INT interrupt is used for START Bit Detect From
                        00097 ;       Idle Mode
                        00098 ;
                        00099 ;*************************************************************************************************
                        00100
0004                    00101 Interrupt:
                        00102 ;
                        00103 ;  At first check if START Bit Detect (currently only INT pin interrupt is enabled, if other
                        00104 ;       Interrupts are enabled, then check for other interrupts)
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
                00105 ;
0004 1C8B       00106         btfss   INTCON,INTF
0005 0009       00107         retfie                          ; other interrupt, simply return & enable GIE
                00108 ;
                00109 ; Save Status
                00110 ;
0006 0099       00111         movwf   SaveW register
0007 0E03       00112         swapf   STATUS,W                ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
0008 0098       00113         movwf   SaveStatus
                00114 ;
0009 1283       00115         bcf     STATUS,RP0
000A 1C06       00116         btfss   _SCL          ; Most likely a START Bit, Hold Time of START Bit must be valid from an INT to here
000B 280F       00117         goto    RestoreStatus           ; Not a Valid START Bit
000C 1886       00118         btfsc   _SDA                    ; If a valid Falling Edge on SDA when SCL is high, SDA must now be Low
000D 280F       00119         goto    RestoreStatus           ; Not a Valid START Bit
000E 282A       00120         goto    StartBitDetect          ; A Valid Start Bit Is Detected, process & then Branch to "Restore Status"
                00121 ;
                00122 ; Restore Status
                00123 ;
000F            00124 RestoreStatus:
                00125
000F 0E18       00126         swapf   SaveStatus,W
0010 0083       00127         movwf   STATUS                  ; restore STATUS Reg
0011 0E99       00128         swapf   SaveW register, F       ; save W register
0012 0E19       00129         swapf   SaveW register,W        ; restore W register
                00130 ;
0013 108B       00131         bcf     INTCON,INTF
0014 0009       00132         retfie
                00133 ;
                00134
                00135 ;*****************************************************************************************************
                00136 ;
                00137 ;*****************************************************************************************************
                00138
0015            00139 Start:
                00140
0015 2019       00141         call    Init_I2C_Slave          ; Initialize I2C Bus for Slave Mode, wait for START Bit detect
                00142 ;
0016 178B       00143         bsf     INTCON,GIE              ; Enable Interrupts
0017            00144 wait:
0017 0064       00145         clrwdt                          ; User can write code here, will be interrupted by Falling Edge on INT
0018 2817       00146         goto    wait                    ; Loop until Start Bit Detect
                00147
                00148
                00149 ;
                00150 ;*****************************************************************************************************
                00151 ;
```

```
                    00152
0019                00153 Init_I2C_Slave:
                    00154
0019 01A0           00155                clrf    ControlByte
001A 1683           00156                bsf     STATUS,RP0
001B 30C3           00157                movlw   _OPTION_INIT
001C 0081           00158                movwf   OPTION_REG      ; initially set INT for Falling Edge INT
001D 1406           00159                bsf     _SCL
001E 1486           00160                bsf     _SDA            ; set TRIS of SCL & SDA to inputs, pulled up by external resistors
                    00161
001F 1283           00162                bcf     STATUS,RP0
0020 0806           00163                movf    PORTB,W
0021 39FC           00164                andlw   0xFC            ; do not use BSF & BCF on Port Pins
0022 0086           00165                movwf   PORTB           ; set SDA & SCL to zero. From Now on, simply play with tris
                    00166
0023 0194           00167                clrf    SubAddr         ; Set Sub Address to Zero
                    00168 ;
                    00169 ; Initialize A/D Setup or EEPROM Control Regs
                    00170 ;
                    00171
                    00172 ;#if AtoD_Slave
                    00173 ;              movlw   0x01            ; Select Channel 0, and turn on A/D Module with Fosc/2 Sampling Rate
                    00174 ;              movwf   _adcon0         ; already in Bank0, STATUS,RP0 = 0
                    00175 ;              bsf     STATUS,RP0
                    00176 ;              clrf   _adcon1          ; 4 analog channels with internal Vref
                    00177 ;#else
0024 0189           00178                clrf    _eeadr          ; already in Bank0, init EEPROM Addr = 0
0025 1683           00179                bsf     STATUS,RP0
0026 0188           00180                clrf    _eecon1         ; clear err flag
                    00181 ;#endif
                    00182 ;
0027 018B           00183                clrf    INTCON
0028 160B           00184                bsf     INTCON,INTE     ; Enable Falling Edge Interrupt On SDA (connected on RB0/INT pin)
0029 0008           00185                return
                    00186
                    00187 ;
                    00188 ;*****************************************************************************************************
                    00189 ;                       In-Line Code For I2C-Slave
                    00190 ;  Returns to detect next Start Bit after a STOP Bit Is Detected
                    00191 ;  This implementation is very in-efficient if the Master is constantly sending a Repeated START Condition
                    00192 ;
                    00193 ;*****************************************************************************************************
                    00194
002A                00195 StartBitDetect:
002A 2045           00196                call    RcvByte         ; 1st byte received in DataByte
002B 0064           00197                clrwdt
002C 1820           00198                btfsc   _STOP
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
002D 283F          00199                goto    i2c_start_wait  ; STOP bit Detected, wait for another START
002E 18A0          00200                btfsc   _START
002F 282A          00201                goto    StartBitDetect  ; a Repeated START condition
                   00202 ;
                   00203 ; The received byte in DataByte contains R/W & 7 Address the address
                   00204 ; If address match send ACK bit else NACK
                   00205 ;
0030 1120          00206                bcf     _RW
0031 180E          00207                btfsc   DataByte,LSB
0032 1520          00208                bsf     _RW             ; LSB of the 1st byte received contains R/W info
0033 100E          00209                bcf     DataByte,LSB
                   00210
0034 080E          00211                movf    DataByte,W
0035 3AD6          00212                xorlw   _MY_ADDRESS     ; if match, then Z bit is set
0036 1D03          00213                btfss   STATUS,Z
0037 283D          00214                goto    SendNACK        ; No Address Match, NACK
                   00215 ;
                   00216 ; Address Match Occured, send ACK
                   00217
0038 1403          00218                bsf     STATUS,C        ; SendAck routine sends ACK if Carry == 1
0039 20B3          00219                call    SendAck
003A 1920          00220                btfsc   _RW
003B 287B          00221                goto    SendReqData     ; read operation, so send current Data
003C 2867          00222                goto    RcvReqData      ; receive 2 bytes of data, sub-addr & data byte
                   00223 ;
                   00224 ;*************************************************************************************************
                   00225 ;
003D               00226 SendNACK:
003D 1003          00227                bcf     STATUS,C        ; SendAck routine sends NACK if Carry == 0
003E 20B3          00228                call    SendAck         ; address not us, so wait for another start bit
                   00229 ;
003F               00230 i2c_start_wait:
003F 0064          00231                clrwdt
0040 1683          00232                bsf     STATUS,RP0
0041 108B          00233                bcf     INTCON,INTF
0042 1406          00234                bsf     _SCL            ; release CLK line, may be held in low by us
0043 1486          00235                bsf     _SDA            ; release SDA
0044 280F          00236                goto RestoreStatus
                   00237 ;
                   00238 ;*************************************************************************************************
                   00239 ;                                Receive A Byte Of Data
                   00240 ;*************************************************************************************************
                   00241
0045               00242 RcvByte:
0045 01A1          00243                clrf    SlvStatus
                   00244
0046 3008          00245                movlw   0x08
```

```
0047 008F          00246                movwf   BitCount
                   00247 ;
0048 1283          00248                bcf     STATUS,RP0
0049 1806          00249                btfsc   _SCL
004A 2849          00250                goto    $-1             ; wait until SCL is low and then Read the Control Byte
                   00251 ;
004B 1683          00252                bsf     STATUS,RP0
004C 1406          00253                bsf     _SCL            ; release CLK, possibly held low
004D 1283          00254                bcf     STATUS,RP0
                   00255 ;
004E               00256 RcvNextBit:
004E 1C06          00257                btfss   _SCL
004F 284E          00258                goto    $-1             ; wait until clock is high, for valid data
0050 1886          00259                btfsc   _SDA
0051 285A          00260                goto    Rcvd_1
0052               00261 Rcvd_0:
0052 1003          00262                bcf     STATUS,C
0053 0D8E          00263                rlf     DataByte, F     ; left shift data ( MSB first)
0054               00264 _WaitClkLo1:
0054 1C06          00265                btfss   _SCL
0055 2862          00266                goto    next1
0056 1C86          00267                btfss   _SDA            ; SDA must still be low when CLK is high, else may be a STOP
0057 2854          00268                goto    _WaitClkLo1
0058 1420          00269                bsf     _STOP
0059 0008          00270                return
                   00271 ;
005A               00272 Rcvd_1:
005A 1403          00273                bsf     STATUS,C
005B 0D8E          00274                rlf     DataByte, F
005C               00275 _WaitClkLo2:
005C 1C06          00276                btfss   _SCL
005D 2862          00277                goto    next1         ; CLK went low, process next bit
005E 1886          00278                btfsc   _SDA          ; SDA must still be high when CLK is high, else may be a Repeated START
005F 285C          00279                goto    _WaitClkLo2
0060 14A0          00280                bsf     _START
0061 0008          00281                return
                   00282 ;
0062               00283 next1:
0062 0B8F          00284                decfsz  BitCount, F
0063 284E          00285                goto    RcvNextBit
                   00286 ;
                   00287 ; A complete Byte Received, HOLD Master's Clock Line Low to force a wait state
                   00288 ;
0064 1683          00289                bsf     STATUS,RP0
0065 1006          00290                bcf     _SCL            ; force SCL Low for wait state
0066 0008          00291                return
                   00292 ;*************************************************************************************************
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

```
                00293 ;                       Write Operation Requested
                00294 ;
                00295 ; Read Sub Address and A Data Byte & Acknowledge, if no errors
                00296 ; Currently Only One Byte Is Programmed At A Time, Buffering scheme is unimplemented
                00297 ;
                00298 ;*************************************************************************************************
                00299
0067            00300 RcvReqData
0067 2045       00301                 call    RcvByte         ;  Sub-Address Byte Received in DataByte, store in Sub-Addr
0068 080E       00302                 movf    DataByte,W
0069 0094       00303                 movwf   SubAddr
006A 0064       00304                 clrwdt
006B 1820       00305                 btfsc   _STOP
006C 283F       00306                 goto    i2c_start_wait  ; STOP bit Detected, wait for another START
006D 18A0       00307                 btfsc   _START
006E 282A       00308                 goto    StartBitDetect  ; a Repeated START condition
006F 1403       00309                 bsf     STATUS,C        ; SendAck routine sends ACK if Carry == 1
0070 20B3       00310                 call    SendAck         ; Sub-Addr Received, Send an ACK
                00311 ;
                00312 ; Receive Data Byte to Write To EEPROM
                00313 ;
0071 2045       00314                 call    RcvByte         ;  Sub-Address Byte Received in DataByte, store in Sub-Addr
0072 0064       00315                 clrwdt
0073 1820       00316                 btfsc   _STOP
0074 283F       00317                 goto    i2c_start_wait  ; STOP bit Detected, wait for another START
0075 18A0       00318                 btfsc   _START
0076 282A       00319                 goto    StartBitDetect  ; a Repeated START condition
0077 1403       00320                 bsf     STATUS,C        ; SendAck routine sends ACK if Carry == 1
0078 20B3       00321                 call    SendAck         ; Sub-Addr Received, Send an ACK
0079 20CD       00322                 call    EEpromWrite     ; Program EEPROM
007A 280F       00323                 goto    RestoreStatus   ; STOP bit Detected, wait for another START
                00324
                00325 ;*************************************************************************************************
                00326 ;                               Read Operation Requested
                00327 ;
                00328 ; Send A/D Data Of Required Channel until NACK By Master
                00329 ;
                00330 ;*************************************************************************************************
                00331
007B            00332 SendReqData:
                00333
007B 1283       00334                 bcf     STATUS,RP0
007C 0814       00335                 movf    SubAddr,W
007D 0089       00336                 movwf   _eeadr          ; Load Sub_Addr Of EEPROM
                00337
                00338 ;
007E            00339 SendNextByte:
```

```
007E 1683          00340                bsf     STATUS,RP0
007F 1408          00341                bsf     _rd             ; read EEPROM
0080 1283          00342                bcf     STATUS,RP0
0081 0808          00343                movf    _eedata,W
0082 008E          00344                movwf   DataByte        ; DataByte = EEPROM(addr)
0083 0A89          00345                incf    _eeadr, F       ; auto-increment sub-address
                   00346
0084 208D          00347                call    TxmtByte        ; send out EEPROM Data
0085 0064          00348                clrwdt
0086 18A0          00349                btfsc   _START          ; check for abnormal START
0087 283F          00350                goto    i2c_start_wait
0088 20C0          00351                call    ReadACK
0089 0064          00352                clrwdt
008A 1DA0          00353                btfss   _ACK           ; _ACK == 1 if +ve ACK Rcvd
008B 283F          00354                goto    i2c_start_wait ; NACK Received, a START or STOP condition may occur
008C 287E          00355                goto    SendNextByte   ; continue to send until NACK
                   00356 ;
                   00357 ;*****************************************************************************************************
                   00358
008D               00359 TxmtByte:
008D 080E          00360                movf    DataByte,W
008E 0093          00361                movwf   DataByteCopy    ; make copy of DataByte
008F 3008          00362                movlw   0x08
0090 008F          00363                movwf   BitCount
0091 01A1          00364                clrf    SlvStatus
                   00365 ;
0092               00366 TxmtNextBit:
0092 1683          00367                bsf     STATUS,RP0
0093 0D93          00368                rlf     DataByteCopy, F ; MSB First
                   00369
0094 1803          00370                btfsc   STATUS,C
0095 28A1          00371                goto    Txmt_1
0096               00372 Txmt_0
0096 1086          00373                bcf     _SDA
0097 0000          00374                nop
0098 1406          00375                bsf     _SCL            ; release clock line, let master pull it high
0099 1283          00376                bcf     STATUS,RP0
009A 1C06          00377                btfss   _SCL
009B 289A          00378                goto    $-1             ; wait until clk goes high
009C 1806          00379                btfsc   _SCL
009D 289C          00380                goto    $-1             ; wait until clk goes low
009E 1683          00381                bsf     STATUS,RP0
009F 1006          00382                bcf     _SCL            ; clk went low, continue to hold it low
00A0 28AD          00383                goto    Q_TxmtNextBit
                   00384
                   00385
00A1               00386 Txmt_1
```


© 1997 Microchip Technology Inc.

```
00A1 1486          00387                bsf     _SDA
00A2 0000          00388                nop
00A3 1406          00389                bsf     _SCL            ; release clock line, let master pull it high
00A4 1283          00390                bcf     STATUS,RP0
00A5 1C06          00391                btfss   _SCL
00A6 28A5          00392                goto    $-1             ; wait until clk goes high
00A7               00393 _IsClkLo_1
00A7 1C86          00394                btfss   _SDA
00A8 28B1          00395                goto    MayBeErr_Txmt   ; must never come here, illegal Repeated Start
00A9 1806          00396                btfsc   _SCL
00AA 28A7          00397                goto    _IsClkLo_1      ; wait until clk goes low
                   00398
00AB 1683          00399                bsf     STATUS,RP0
00AC 1006          00400                bcf     _SCL            ; clk went low, continue to hold it low
                   00401 ;
00AD               00402 Q_TxmtNextBit
00AD 0B8F          00403                decfsz  BitCount, F
00AE 2892          00404                goto    TxmtNextBit
00AF 1486          00405                bsf     _SDA            ; release SDA for Master's ACK
00B0 0008          00406                return
                   00407 ;
00B1               00408 MayBeErr_Txmt:
00B1 14A0          00409                bsf     _START          ; illegal Repeated START condition during a byte transfer
00B2 0008          00410                return
                   00411 ;
                   00412 ;*****************************************************************************************************
                   00413 ;                              Send ACK/NACK to Master
                   00414 ;
                   00415 ;  Prior to calling this routine, set CARRY bit to 1 for sending +ve ACK & set CARRY = 0, for NACK
                   00416 ;
                   00417 ;*****************************************************************************************************
                   00418
00B3               00419 SendAck:
00B3 1683          00420                bsf     STATUS,RP0
00B4 1803          00421                btfsc   STATUS,C        ; Carry bit == 1 for ACK else NACK
00B5 1086          00422                bcf     _SDA            ; pull SDA low to send +ve ACK
00B6 1406          00423                bsf     _SCL            ; release CLK line, let master clk it
00B7 1283          00424                bcf     STATUS,RP0
00B8 1C06          00425                btfss   _SCL
00B9 28B8          00426                goto    $-1             ; loop until Clk High
00BA 1806          00427                btfsc   _SCL
00BB 28BA          00428                goto    $-1             ; loop until Clk is Low, ACK bit sent
00BC 1683          00429                bsf     STATUS,RP0
00BD 1006          00430                bcf     _SCL            ; HOLD CLK Line LOW
00BE 1486          00431                bsf     _SDA            ; ACK over, release SDA line for Master control
                   00432 ;
                   00433 ;
```

© 1997 Microchip Technology Inc.

```
00BF 0008          00434                return
                   00435 ;
                   00436 ;*****************************************************************************************************
                   00437 ;                                    Read ACK Sent By Master
                   00438 ;
                   00439 ; If +ve ACK then set _ACK bit in SlaveStatus Reg, else 0
                   00440 ;
                   00441 ;*****************************************************************************************************
                   00442
00C0               00443 ReadACK:
00C0 1683          00444                bsf     STATUS,RP0
00C1 1406          00445                bsf     _SCL            ; release clock
00C2 1283          00446                bcf     STATUS,RP0
00C3 1C06          00447                btfss   _SCL
00C4 28C3          00448                goto    $-1             ; wait until clock is high (9 the bit:ACK)
00C5 15A0          00449                bsf     _ACK            ; expecting a +ve ACK
00C6 1886          00450                btfsc   _SDA
00C7 11A0          00451                bcf     _ACK            ; NACK rcvd, stop transmission
00C8 1806          00452                btfsc   _SCL
00C9 28C8          00453                goto    $-1             ; wait until Clock is low
00CA 1683          00454                bsf     STATUS,RP0
00CB 1006          00455                bcf     _SCL            ; force Clock low
00CC 0008          00456                return
                   00457 ;*****************************************************************************************************
                   00458 ;                    Write One Byte Of Data To EEPROM Array at Sub-Addr
                   00459 ;
                   00460 ;*****************************************************************************************************
00CD               00461 EEpromWrite:
00CD 1283          00462                bcf     STATUS,RP0
00CE 0814          00463                movf    SubAddr,W
00CF 0089          00464                movwf   _eeadr          ; load sub-address
00D0 080E          00465                movf    DataByte,W
00D1 0088          00466                movwf   _eedata         ; load data to be written into EEDATA
00D2 1683          00467                bsf     STATUS,RP0
00D3 1508          00468                bsf     _wren           ; enable write operation
00D4 1188          00469                bcf     _wrerr          ; clear any previous error flags
00D5 3055          00470                movlw   0x55
00D6 0089          00471                movwf   _eecon2
00D7 30AA          00472                movlw   0xAA
00D8 0089          00473                movwf   _eecon2
00D9 1488          00474                bsf     _wr             ; start Write Operation
                   00475 ;
00DA 1406          00476                bsf     _SCL            ; release CLK line, may be held in low by us
00DB 1486          00477                bsf     _SDA            ; release SDA
                   00478 ;
00DC 0064          00479                clrwdt
00DD 1888          00480                btfsc   _wr             ; Poll until WR Over
```


© 1997 Microchip Technology Inc.

```
00DE 28DC           00481                goto    $-2
                    00482 ;
00DF 1108           00483                bcf     _wren           ; disable write operations
00E0 0064           00484                clrwdt
00E1 108B           00485                bcf     INTCON,INTF
                    00486 ;
00E2 0008           00487                return
                    00488 ;*************************************************************************************************************
                    00489
                    00490
                    00491     end

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)


0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXX------------ ----------------

All other memory blocks unused.

Program Memory Words Used:   224
Program Memory Words Free:   800


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     1 suppressed
```

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: TEST.ASM

```
MPASM 01.40.01 Intermediate   I2CTEST.ASM   4-4-1997  14:45:51         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         Title           "I2C Master Mode Implemetation"
                     00002         SubTitle        "Rev 0.2        :  04 April 1997"
                     00003
                     00004 ;*******************************************************************************************************
                     00005 ;
                     00006 ;                       Software Implementation Of I2C Master Mode
                     00007 ;
                     00008 ;       * Master Transmitter & Master Receiver Implemented in software
                     00009 ;       * Slave Mode implemented in hardware
                     00010 ;
                     00011 ;       *       Refer to Signetics/Philips I2C-Bus Specification
                     00012 ;
                     00013 ;    The software is implemented using PIC16C71 & thus can be ported to all Enhanced core PIC16CXX product
                     00014 ;
                     00015 ;       RB1 is SDA              (Any I/O Pin May Be used instead)
                     00016 ;       RB0/INT is SCL          (Any I/O Pin May Be used instead)
                     00017 ;
                     00018 ;
                     00019 ;       Program:          I2CTEST.ASM
                     00020 ;       Revision Date:    Rev 0.1       :  01 Mar 1993
                     00021 ;                         4-04-97      Compatibility with MPASMWIN 1.40
                     00022 ;
                     00023 ;*******************************************************************************************************
                     00024
                     00025                 LIST    p = 16C71
                     00026                 ERRORLEVEL -302
                     00027                 Radix   DEC
                     00028
  00F42400           00029 _ClkIn          equ     16000000        ; Input Clock Frequency Of PIC16C71
                     00030
                     00031                 include         <p16c71.inc>
                     00001         LIST
                     00002 ; P16C71.INC  Standard Header File, Version 1.00    Microchip Technology, Inc.
                     00142         LIST
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
                        00032
00000001                00033 TRUE    equ     1
00000000                00034 FALSE   equ     0
                        00035
00000000                00036 LSB     equ     0
00000007                00037 MSB     equ     7
                        00038
                        00039 ;
                        00040 #define _Slave_1_Addr   0xA0            ; Serial EEPROM #1
                        00041 #define _Slave_2_Addr   0xAC            ; Serial EEPROM #2
                        00042 #define _Slave_3_Addr   0xD6            ; Slave PIC16CXX
                        00043
                        00044 #define _ENABLE_BUS_FREE_TIME   TRUE
                        00045 #define _CLOCK_STRETCH_CHECK    TRUE
                        00046 #define _INCLUDE_HIGH_LEVEL_I2C TRUE
                        00047
                        00048
                        00049                 include         "i2c.h"
                        00001 ;*****************************************************************************************************
                        00002 ;                               I2C Bus Header File
                        00003 ;*****************************************************************************************************
                        00004
003D0900                00005 _ClkOut         equ     (_ClkIn >> 2)
                        00006
                        00007 ;
                        00008 ; Compute the delay constants for setup & hold times
                        00009 ;
00000010                00010 _40uS_Delay     set     (_ClkOut/250000)
00000012                00011 _47uS_Delay     set     (_ClkOut/212766)
00000014                00012 _50uS_Delay     set     (_ClkOut/200000)
                        00013
                        00014 #define _OPTION_INIT    (0xC0 | 0x03)   ; Prescaler to TMR0 for Appox 1 mSec timeout
                        00015 ;
                        00016 #define _SCL    PORTB,0
                        00017 #define _SDA    PORTB,1
                        00018
                        00019 #define _SCL_TRIS       _trisb,0
                        00020 #define _SDA_TRIS       _trisb,1
                        00021
                        00022 #define _WRITE_         0
                        00023 #define _READ_          1
                        00024
                        00025
                        00026                                   ; Register File Variables
                        00027
                        00028
                        00029                 CBLOCK  0x0C
```

```
0000000C 00030                       SlaveAddr              ; Slave Addr must be loaded into this reg
0000000D 00031                       SlaveAddrHi            ; for 10 bit addressing mode
0000000E 00032                       DataByte               ; load this reg with the data to be transmitted
0000000F 00033                       BitCount               ; The bit number (0:7) transmitted or received
00000010 00034                       Bus_Status             ; Status Reg of I2C Bus for both TXMT & RCVE
00000011 00035                       Bus_Control            ; control Register of I2C Bus
00000012 00036                       DelayCount
00000013 00037                       DataByteCopy           ; copy of DataByte for Left Shifts (destructive)
         00038
00000014 00039                       SubAddr                ; sub-address of slave (used in I2C_HIGH.ASM)
00000015 00040                       SrcPtr                 ; source pointer for data to be transmitted
         00041
00000016 00042                       tempCount              ; a temp variable for scratch RAM
00000017 00043                       StoreTemp_1            ; a temp variable for scratch RAM, do not disturb contents
         00044
00000018 00045                       _End_I2C_Ram           ; unused, only for ref of end of RAM allocation
         00046                 ENDC
         00047
         00048 ;*************************************************************************************
         00049 ;                     I2C Bus Status Reg Bit Definitions
         00050 ;*************************************************************************************
         00051
         00052 #define _Bus_Busy       Bus_Status,0
         00053 #define _Abort          Bus_Status,1
         00054 #define _Txmt_Progress  Bus_Status,2
         00055 #define _Rcv_Progress   Bus_Status,3
         00056
         00057 #define _Txmt_Success   Bus_Status,4
         00058 #define _Rcv_Success    Bus_Status,5
         00059 #define _Fatal_Error    Bus_Status,6
         00060 #define _ACK_Error      Bus_Status,7
         00061
         00062 ;*************************************************************************************
         00063 ;                     I2C Bus Contro Register
         00064 ;*************************************************************************************
         00065 #define _10BitAddr      Bus_Control,0
         00066 #define _Slave_RW       Bus_Control,1
         00067 #define _Last_Byte_Rcv  Bus_Control,2
         00068
         00069 #define _SlaveActive    Bus_Control,6
         00070 #define _TIME_OUT_      Bus_Control,7
         00071
         00072
         00073
         00074
         00075 ;*****************************************************************************************************
         00076 ;                              General Purpose Macros
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
                        00077 ;*****************************************************************************************************
                        00078
                        00079 RELEASE_BUS     MACRO
                        00080                         bsf     STATUS,RP0      ; select Bank1
                        00081                         bsf     _SDA            ; tristate SDA
                        00082                         bsf     _SCL            ; tristate SCL
                        00083 ;                       bcf     _Bus_Busy       ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
                        00084                 ENDM
                        00085
                        00086 ;*****************************************************************************************************
                        00087 ;                       A MACRO To Load 8 OR 10 Bit Address To The Address Registers
                        00088 ;
                        00089 ;  SLAVE_ADDRESS is a constant and is loaded into the SlaveAddress Register(s) depending
                        00090 ;  on 8 or 10 bit addressing modes
                        00091 ;*****************************************************************************************************
                        00092
                        00093 LOAD_ADDR_10    MACRO   SLAVE_ADDRESS
                        00094
                        00095                 bsf     _10BitAddr      ; Slave has 10 bit address
                        00096                 movlw   (SLAVE_ADDRESS & 0xff)
                        00097                 movwf   SlaveAddr       ; load low byte of address
                        00098                 movlw   (((SLAVE_ADDRESS >> 7) & 0x06) | 0xF0)     ; 10 bit addr is 11110XX0
                        00099                 movwf   SlaveAddr+1     ; hi order  address
                        00100
                        00101                 ENDM
                        00102
                        00103 LOAD_ADDR_8     MACRO   SLAVE_ADDRESS
                        00104
                        00105                 bcf     _10BitAddr      ; Set for 8 Bit Address Mode
                        00106                 movlw   (SLAVE_ADDRESS & 0xff)
                        00107                 movwf   SlaveAddr
                        00108
                        00109                 ENDM
                        00050
                        00051                 CBLOCK  _End_I2C_Ram
00000018                00052                         SaveStatus              ; copy of STATUS Reg
00000019                00053                         SaveW register          ; copy of W register
0000001A                00054                         byteCount
0000001B                00055                         HoldData
                        00056                 ENDC
                        00057
                        00058                 CBLOCK  0x20
00000020                00059                         DataBegin               ; Data to be read or written is stored here
                        00060                 ENDC
                        00061
                        00062
0000                    00063                 ORG     0x00
```

```
                        00064
0000 2956               00065                   goto    Start
                        00066 ;
0004                    00067                   ORG     0x04
                        00068 ;*****************************************************************************************************
                        00069 ;                                 Interrupt Service Routine
                        00070 ;
                        00071 ;   For I2C routines, only TMR0 interrupt is used
                        00072 ; TMR0 Interrupts enabled only if Clock Stretching is Used
                        00073 ; On TMR0 timeout interrupt, disable TMR0 Interrupt, clear pending flags,
                        00074 ; MUST set _TIME_OUT_ flag saying possibly a FATAL error occurred
                        00075 ; The user may choose to retry the operation again later
                        00076 ;
                        00077 ;*****************************************************************************************************
                        00078
0004                    00079 Interrupt:
                        00080 ;
                        00081 ; Save Interrupt Status (W register & STATUS regs)
                        00082 ;
0004 0099               00083           movwf   SaveW register          ; Save W register
0005 0E03               00084           swapf   STATUS,W                ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
0006 0098               00085           movwf   SaveStatus              ; Save STATUS Reg
                        00086  if _CLOCK_STRETCH_CHECK                  ; TMR0 Interrupts enabled only if Clock Stretching is Used
0007 1D0B               00087           btfss   INTCON,T0IF
0008 280B               00088           goto    MayBeOtherInt           ; other Interrupts
0009 1791               00089           bsf     _TIME_OUT_              ; MUST set this Flag, can take other desired actions here
000A 110B               00090           bcf     INTCON,T0IF
                        00091  endif
                        00092 ;
                        00093 ;  Check For Other Interrupts Here, This program usesd only TMR0 & INT Interrupt
                        00094 ;
000B                    00095 MayBeOtherInt:
000B 0000               00096           NOP
                        00097 ;
000C                    00098 RestoreIntStatus:                         ; Restore Interrupt Status
000C 0E18               00099           swapf   SaveStatus,W
000D 0083               00100           movwf   STATUS                  ; restore STATUS Reg
000E 0E99               00101           swapf   SaveW register, F
000F 0E19               00102           swapf   SaveW register,W        ; restore W register
0010 0009               00103           retfie
                        00104 ;
                        00105 ;*****************************************************************************************************
                        00106 ;                                 Include I2C High Level & Low Level Routines
                        00107   if _INCLUDE_HIGH_LEVEL_I2C
                        00108                   include         "i2c_high.inc"
                        00001 ;*****************************************************************************************************
                        00002 ;
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
00003 ;                              I2C Master : General Purpose Macros & Subroutines
00004 ;
00005 ;               High Level Routines, Uses Low level Routines (in I2C_LOW.ASM)
00006 ;
00007 ;
00008 ;       Program:          I2C_HIGH.ASM
00009 ;       Revision Date:
00010 ;                         1-16-97      Compatibility with MPASMWIN 1.40
00011 ;
00012 ;*******************************************************************************************************
00013
00014
00015 ;*******************************************************************************************************
00016 ;
00017 ;                                I2C_TEST_DEVICE
00018 ; MACRO
00019 ;
00020 ;   If Slave Device is listening, then _SlaveActive bit is set, else is cleared
00021 ;
00022 ;       Parameter :  NONE
00023 ;
00024 ;   Sequence Of Operations :
00025 ;       S-SlvAW-A-P
00026 ;   If A is +ve device is listening, else either busy, not present or error condition
00027 ;
00028 ;   This test may also be used to check for eample if a Serial EEPROM is in internal programming
00029 ;   mode
00030 ;
00031 ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
00032 ;         mode addressing must be set
00033 ;*******************************************************************************************************
00034
00035 I2C_TEST_DEVICE         MACRO
00036
00037                         call    IsSlaveActive           ; TEMP ???? : Assembler Error with this MACRO
00038
00039                         ENDM
00040 ;
00041 ;
00042 ;               Test If A Device of SlaveAddr Is Present on Bus
00043 ;
00044 ;  The Slave Address Is put on the bus and if ACK it is present, if NACK not present
00045 ;  or maybe device is not responding. The presense can be checked constantly by a master
00046 ;  (for ex. the Operating System on an Access.Bus may constantly issue this command)
00047 ;
00048 ;   Assume the Slave Address (10 or 8 bit) is loaded in SlaveAddr
00049 ;   Set  _10BitAddr bit in Control Reg to 1 if 10 bit Address slave else 0
```

```
                00050 ;
                00051 ;   Returns  1 in _SlaveActive Bit if slave is responding else a 0
                00052 ;
                00053 ;
                00054
0011            00055 IsSlaveActive
0011 1091       00056                 bcf     _Slave_RW       ; set for write operation
0012 2057       00057                 call    TxmtStartBit    ; send START bit
0013 206B       00058                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
                00059 ;
0014 1311       00060                 bcf     _SlaveActive
0015 1F90       00061                 btfss   _ACK_Error      ; skip if NACK, device is not present or not responding
0016 1711       00062                 bsf     _SlaveActive    ; ACK received, device present & listening
0017 205F       00063                 call    TxmtStopBit
0018 0008       00064                 return
                00065 ;
                00066 ;*****************************************************************************************************
                00067 ;                               I2C_WRITE
                00068 ;
                00069 ;  A basic macro for writing a block of data to a slave
                00070 ;
                00071 ;  Parameters :
                00072 ;               _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_
                00073 ;               _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
                00074 ;
                00075 ;   Sequence :
                00076 ;               S-SlvAW-A-D[0]-A.....A-D[N-1]-A-P
                00077 ;
                00078 ;  If an error occurs then the routine simply returns and user should check for
                00079 ;       flags in Bus_Status Reg (for eg. _Txmt_Success flag)
                00080 ;
                00081 ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
                00082 ;               mode addressing must be set
                00083 ;*****************************************************************************************************
                00084
                00085
                00086 I2C_WR          MACRO   _BYTES_, _SourcePointer_
                00087
                00088                 movlw   _BYTES_
                00089                 movwf   tempCount
                00090                 movlw   _SourcePointer_
                00091                 movwf   FSR
                00092
                00093                 call    _i2c_block_write
                00094                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
                00095
                00096                 ENDM
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
                        00097
0019                    00098 _i2c_block_write:
0019 2057               00099                 call    TxmtStartBit    ; send START bit
001A 1091               00100                 bcf     _Slave_RW       ; set for write operation
001B 206B               00101                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
                        00102 ;
001C                    00103 _block_wr1_loop:
001C 1E10               00104                 btfss   _Txmt_Success
001D 0008               00105                 return
001E 0800               00106                 movf    INDF,W
001F 008E               00107                 movwf   DataByte        ; start from the first byte starting at _DataPointer_
0020 0A84               00108                 incf    FSR, F
0021 2095               00109                 call    SendData        ; send next byte, bus is our's !
0022 0B96               00110                 decfsz  tempCount, F
0023 281C               00111                 goto    _block_wr1_loop ; loop until desired bytes of data transmitted to slave
0024 0008               00112                 return
                        00113 ;
                        00114 ;*****************************************************************************************************
                        00115
                        00116 ;*****************************************************************************************************
                        00117 ;                           I2C_WRITE_SUB
                        00118 ;
                        00119 ;  Writes a message just like I2C_WRITE, except that the data is preceeded by a sub-address
                        00120 ;  to a slave device.
                        00121 ;             Eg. : A serial EEPROM would need an address of memory location for Random Writes
                        00122 ;
                        00123 ;  Parameters :
                        00124 ;               _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_ (constant)
                        00125 ;               _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
                        00126 ;               _Sub_Address_           Sub-address of Slave (constant)
                        00127 ;
                        00128 ;   Sequence :
                        00129 ;               S-SlvAW-A-SubA-A-D[0]-A.....A-D[N-1]-A-P
                        00130 ;
                        00131 ;  If an error occurs then the routine simply returns and user should check for
                        00132 ;  flags in Bus_Status Reg (for eg. _Txmt_Success flag
                        00133 ;
                        00134 ;           Returns :  W register = 1 on success, else W register = 0
                        00135 ;
                        00136 ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
                        00137 ;         mode addressing must be set
                        00138 ;
                        00139 ;  COMMENTS :
                        00140 ;               I2C_WR may prove to be more efficient than this macro in most situations
                        00141 ;               Advantages will be found for Random Address Block Writes for Slaves with
                        00142 ;               Auto Increment Sub-Addresses (like Microchip's 24CXX series Serial EEPROMS)
                        001[illegible] ;
```

```
00144 ;*************************************************************************************************************
00145
00146 I2C_WR_SUB      MACRO   _BYTES_, _SourcePointer_, _Sub_Address_
00147
00148                 movlw   (_BYTES_ + 1)
00149                 movwf   tempCount
00150
00151                 movlw   (_SourcePointer_ - 1)
00152                 movwf   FSR
00153
00154                 movf    INDF,W
00155                 movwf   StoreTemp_1     ; temporarily store contents of (_SourcePointer_ -1)
00156                 movlw   _Sub_Address_
00157                 movwf   INDF            ; store temporarily the sub-address at (_SourcePointer_ -1)
00158
00159                 call    _i2c_block_write        ; write _BYTES_+1 block of data
00160
00161                 movf    StoreTemp_1,W
00162                 movwf   (_SourcePointer_ - 1)   ; restore contents of (_SourcePointer_ - 1)
00163
00164                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
00165
00166                 ENDM
00167
00168 ;*************************************************************************************************************
00169 ;                               I2C_WR_SUB_SWINC
00170 ;
00171 ;  Parameters :
00172 ;               _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_ (constant)
00173 ;               _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
00174 ;               _Sub_Address_           Sub-address of Slave (constant)
00175 ;
00176 ;   Sequence :
00177 ;               S-SlvAW-A-(SubA+0)-A-D[0]-A-P
00178 ;               S-SlvAW-A-(SubA+1)-A-D[1]-A-P
00179 ;                       and so on until #of Bytes
00180 ;
00181 ;  If an error occurs then the routine simply returns and user should check for
00182 ;       flags in Bus_Status Reg (for eg. _Txmt_Success flag
00183 ;
00184 ;       Returns :       W register = 1 on success, else W register = 0
00185 ;
00186 ;  COMMENTS : Very In-efficient, Bus is given up after every Byte Write
00187 ;
00188 ;               Some I2C devices addresed with a sub-address do not increment automatically
00189 ;       after an access of each byte. Thus a block of data sent must have a sub-address
00190 ;       followed by a data byte.
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
                00191 ;
                00192 ;*****************************************************************************************************************
                00193
                00194 I2C_WR_SUB_SWINC        MACRO   _BYTES_, _SourcePointer_, _Sub_Address_
                00195
                00196                        variable i              ; TEMP ???? : Assembler Does Not Support This
                00197
                00198                        i = 0
                00199
                00200                          while (i < _BYTES_)
                00201                                movf    (_Source_Pointer_ + i),W
                00202                                movwf   SrcPtr
                00203                                movf    (_Sub_Address_ + i),W
                00204                                movwf   SubAddr
                00205                                call    _i2c_byte_wr_sub        ; write a byte of data at sub address
                00206
                00207                                i++
                00208                        endw
                00209
                00210                         ENDM
                00211
                00212 ;
                00213 ;
                00214 ;  Write 1 Byte Of Data (in SrcPtr) to slave at sub-address (SubAddr)
                00215 ;
                00216
0025            00217 _i2c_byte_wr_sub:
0025 2057       00218                call    TxmtStartBit    ; send START bit
0026 1091       00219                bcf     _Slave_RW       ; set for write operation
0027 206B       00220                call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
0028 1E10       00221                btfss   _Txmt_Success
0029 2835       00222                goto    _block_wr1_fail ; end
002A 0814       00223                movf    SubAddr,W
002B 008E       00224                movwf   DataByte        ; start from the first byte starting at _DataPointer_
002C 2095       00225                call    SendData        ; send next byte
002D 1E10       00226                btfss   _Txmt_Success
002E 2835       00227                goto    _block_wr1_fail ; end
002F 0815       00228                movf    SrcPtr,W
0030 008E       00229                movwf   DataByte        ; start from the first byte starting at _DataPointer_
0031 2095       00230                call    SendData        ; send next byte
0032 1E10       00231                btfss   _Txmt_Success
0033 2835       00232                goto    _block_wr1_fail ; failed, return 0 in W register
0034 2837       00233                goto    _block_wr1_pass ; successful, return 1 in W register
                00234 ;
                00235 ; return back to called routine from either _block_wr1_pass or _block_wr1_fail
                00236 ;
0035            00237 _block_wr1_fail:
```

```
0035 205F          00238                call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
0036 3400          00239                retlw   FALSE
0037               00240 _block_wr1_pass:
0037 205F          00241                call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
0038 3401          00242                retlw   TRUE
                   00243 ;
                   00244
                   00245 ;*****************************************************************************************************
                   00246 ;
                   00247 ;                              I2C_WR_MEM_BYTE
                   00248 ;
                   00249 ;  Some I2C devices like a EEPROM need to wait fo some time after every byte write
                   00250 ;  (when entered into internal programming mode). This MACRO is same as I2C_WR_SUB_SWINC,
                   00251 ;  but in addition adds a delay after each byte.
                   00252 ;          Some EERPOM memories (like Microchip's 24Cxx Series have on-chip data buffer), and hence
                   00253 ;               this routine is not efficient in these cases. In such cases use I2C_WR or I2C_WR_SUB
                   00254 ;               for a block of data and then insert a delay until the whole buffer is written.
                   00255 ;
                   00256 ;  Parameters :
                   00257 ;               _BYTES_                 #of bytes starting from RAM pointer _SourcePointer_ (constant)
                   00258 ;               _SourcePointer_         Data Start Buffer pointer in RAM (file Registers)
                   00259 ;               _Sub_Address_           Sub-address of Slave (constant)
                   00260 ;
                   00261 ;   Sequence :
                   00262 ;               S-SlvAW-A-(SubA+0)-A-D[0]-A-P
                   00263 ;               Delay 1 mSec            ; The user can chnage this value to desired delay
                   00264 ;               S-SlvAW-A-(SubA+1)-A-D[1]-A-P
                   00265 ;               Delay 1 mSec
                   00266 ;               and so on until #of Bytes
                   00267 ;
                   00268 ;*****************************************************************************************************
                   00269
                   00270 I2C_WR_BYTE_MEM         MACRO   _BYTES_, _SourcePointer_, _Sub_Address_
                   00271
                   00272                       variable i              ; TEMP ???? : Assembler Does Not Support This
                   00273
                   00274                       i = 0
                   00275
                   00276                       while (i < _BYTES_)
                   00277                               movf    (_Source_Pointer_ + i),W
                   00278                               movwf   SrcPtr
                   00279                               movf    (_Sub_Address_ + i),W
                   00280                               movwf   SubAddr
                   00281                               call    _i2c_byte_wr_sub        ; write a byte of data at sub address
                   00282                               call    Delay50uSec
                   00283
                   00284                               i++
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
00285                       endw
00286
00287
00288                        ENDM
00289
00290 ;*****************************************************************************************************************
00291 ;                   I2C_WR_MEM_BUF
00292 ;
00293 ;   This Macro/Function writes #of _BYTES_ to an I2C memory device. However
00294 ;   some devices, esp. EEPROMs must wait while the device enters into programming
00295 ;   mode. But some devices have an onchip temp data hold buffer and is used to
00296 ;   store data before the device actually enters into programming mode.
00297 ;   For example, the 24C04 series of Serial EEPROMs from Microchip
00298 ;   have an 8 byte data buffer. So one can send 8 bytes of data at a time
00299 ;   and then the device enters programming mode. The master can either wait
00300 ;   until a fixed time and then retry to program or can continiously poll
00301 ;   for ACK bit and then transmit the next Block of data for programming
00302 ;
00303 ;  Parameters :
00304 ;               _BYTES_                 # of bytes to write to memory
00305 ;               _SourcePointer_         Pointer to the block of data
00306 ;               _SubAddress_            Sub-address of the slave
00307 ;               _Device__BUF_SIZE_      The on chip buffer size of the i2c slave
00308 ;
00309 ;  Sequence of operations
00310 ;                       I2C_SUB_WR operations are performed in loop and each time
00311 ;                       data buffer of BUF_SIZE is output to the device. Then
00312 ;                       the device is checked for busy and when not busy another
00313 ;                       block of data is written
00314 ;
00315 ;
00316 ;*****************************************************************************************************************
00317
00318 I2C_WR_BUF_MEM  MACRO   _BYTES_, _SourcePointer_, _SubAddress_, _Device_BUF_SIZE_
00319
00320
00321                variable i, j
00322
00323
00324                if ( !_BYTES_)
00325                        exitm
00326
00327                elif ( _BYTES_ <=  _Device_BUF_SIZE_)
00328
00329                        I2C_WR_SUB      _BYTES_, _SourcePointer_, _SubAddress_
00330
00331                        exitm
```

```
00332
00333                 else
00334
00335                         i = 0
00336                         j = (_BYTES_ / _Device_BUF_SIZE_)
00337                         while (i < j)
00338
00339 I2C_WR_SUB  _Device_BUF_SIZE_, (_SourcePointer_ + i*_Device_BUF_SIZE_),(_SubAddress_ + i*_Device_BUF_SIZE_)
00340
00341                                 call    IsSlaveActive
00342                                 btfss   _SlaveActive
00343                                 goto    $-2
00344
00345                           i++
00346                           endw
00347
00348                         j = (_BYTES_ - i*_Device_BUF_SIZE_)
00349
00350                         if (j)
00351 I2C_WR_SUB j, (_SourcePointer_ + i*_Device_BUF_SIZE_), (_SubAddress_  + i*_Device_BUF_SIZE_)
00352                         endif
00353
00354                 endif
00355
00356                 ENDM
00357
00358
00359 ;*****************************************************************************************************
00360 ;
00361 ;                               I2C_READ
00362 ;
00363 ; The basic MACRO/procedure to read a block message from a slave device
00364 ;
00365 ;   Parameters :
00366 ;               _BYTES_         :  constant : #of bytes to receive
00367 ;               _DestPointer_   :  destination pointer of RAM (File Registers)
00368 ;
00369 ;   Sequence :
00370 ;               S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P
00371 ;
00372 ;   If last byte, then Master will NOT Acknowledge (send NACK)
00373 ;
00374 ;  NOTE : The address of the slave must be loaded into SlaveAddress Registers, and 10 or 8 bit
00375 ;         mode addressing must be set
00376 ;
00377 ;*****************************************************************************************************
00378
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
                00379 I2C_READ        MACRO   _BYTES_, _DestPointer_
                00380
                00381
                00382                 movlw   (_BYTES_ -1)
                00383                 movwf   tempCount               ; -1 because, the last byte is used out of loop
                00384                 movlw   _DestPointer_
                00385                 movwf   FSR                     ; FIFO destination address pointer
                00386
                00387                 call    _i2c_block_read
                00388
                00389                 ENDM
                00390
0039            00391 _i2c_block_read:
0039 2057       00392                 call    TxmtStartBit    ; send START bit
003A 1491       00393                 bsf     _Slave_RW       ; set for read operation
003B 1111       00394                 bcf     _Last_Byte_Rcv  ; not a last byte to rcv
003C 206B       00395                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
003D 1A10       00396                 btfsc   _Txmt_Success
003E 2841       00397                 goto    _block_rd1_loop ; end
003F 205F       00398                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
0040 3400       00399                 retlw   FALSE           ; Error : may be device not responding
                00400 ;
0041            00401 _block_rd1_loop:
0041 20CC       00402                 call    GetData
0042 080E       00403                 movf    DataByte,W
0043 0080       00404                 movwf   INDF            ; start receiving data, starting at Destination Pointer
0044 0A84       00405                 incf    FSR, F
0045 0B96       00406                 decfsz  tempCount, F
0046 2841       00407                 goto    _block_rd1_loop ; loop until desired bytes of data transmitted to slave
0047 1511       00408                 bsf     _Last_Byte_Rcv  ; last byte to rcv, so send NACK
0048 20CC       00409                 call    GetData
0049 080E       00410                 movf    DataByte,W
004A 0080       00411                 movwf   INDF
004B 205F       00412                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
004C 3401       00413                 retlw   TRUE
                00414
                00415
                00416 ;*****************************************************************************************************
                00417 ;
                00418 ;                                   I2C_READ_SUB
                00419 ;  This MACRO/Subroutine reads a message from a slave device preceeded by a write of the sub-address
                00420 ;  Between the sub-address write & the following reads, a STOP condition is not issued and
                00421 ;  a "REPEATED START" condition is used so that an other master will not take over the bus,
                00422 ;  and also that no other master will overwrite the sub-address of the same slave.
                00423 ;
                00424 ;   This function is very commonly used in accessing Random/Sequential reads from a
                00425 ;   memory device (e.g : 24Cxx serial of Serial EEPROMs from Microchip).
```

```
00426 ;
00427 ;  Parameters :
00428 ;               _BYTES_         # of bytes to read
00429 ;               _DestPointer_   The destination pointer of data to be received.
00430 ;               _BubAddress_    The sub-address of the slave
00431 ;
00432 ;  Sequence :
00433 ;               S-SlvAW-A-SubAddr-A-S-SlvAR-A-D[0]-A-.....-A-D[N-1]-N-P
00434 ;
00435 ;
00436 ;*************************************************************************************************************
00437
00438 I2C_READ_SUB    MACRO   _BYTES_, _DestPointer_, _SubAddress_
00439
00440                 bcf     _Slave_RW       ; set for write operation
00441                 call    TxmtStartBit    ; send START bit
00442                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
00443
00444                 movlw   _SubAddress_
00445                 movwf   DataByte        ; START address of EEPROM(slave 1)
00446                 call    SendData        ; write sub address
00447 ;
00448 ; do not send STOP after this, use REPEATED START condition
00449 ;
00450
00451                 I2C_READ _BYTES_, _DestPointer_
00452
00453
00454                 ENDM
00455
00456
00457
00458 ;*************************************************************************************************************
00459 ;
00460 ;                               I2C_READ_STATUS
00461 ;
00462 ;  This Macro/Function reads a status word (1 byte) from slave. Several I2C devices can
00463 ;  send a status byte upon reception of a control byte
00464 ;  This is basically same as I2C_READ MACRO for reading a single byte
00465 ;
00466 ;   For example, in a Serial EEPROM (Microchip's 24Cxx serial EEPROMs) will send the memory
00467 ;   data at the current address location
00468 ;
00469 ;  On success W register = 1 else = 0
00470 ;
00471 ;*************************************************************************************************************
00472
```


© 1997 Microchip Technology Inc.

```
00473 I2C_READ_STATUS MACRO   _DestPointer_
00474
00475
00476                 call    TxmtStartBit    ; send START bit
00477                 bsf     _Slave_RW       ; set for read operation
00478                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
00479                 btfsc   _Txmt_Success
00480                 goto    _byte_rd1_loop  ; read a byte
00481                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
00482                 retlw   FALSE           ; Error : may be device not responding
00483 _byte_rd1_loop:
00484                 bsf     _Last_Byte_Rcv  ; last byte to rcv, so send NACK
00485                 call    GetData
00486                 movf    DataByte,W
00487                 movwf   _DestPointer_
00488                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
00489                 btfss   _Rcv_Success
00490                 retlw   FALSE
00491                 retlw   TRUE
00492
00493                 ENDM
00494
00495 ;*****************************************************************************************************************
00496
00497 I2C_READ_BYTE   MACRO   _DestPointer_
00498
00499                 I2C_READ_STATUS MACRO   _DestPointer_
00500
00501                 ENDM
00502
00503
00504 ;*****************************************************************************************************************
00505 ;
00506 ;                               I2C_WR_SUB_WR
00507 ;
00508 ;  This Macro writes 2 Blocks of Data (variable length) to a slave at a sub-address. This
00509 ;  may be useful for devices which need 2 blocks of data in which the first block may be an
00510 ;  extended address of a slave device. For example, a large I2C memory device, or a teletext
00511 ;  device with an extended addressing scheme, may need multiple bytes of data in the 1st block
00512 ;  that represents the actual physical address and is followed by a 2nd block that actually
00513 ;  represents the data.
00514 ;
00515 ;  Parameters :
00516 ;
00517 ;               _BYTES1_                1st block #of bytes
00518 ;               _SourcePointer1_        Start Pointer of the 1st block
00519 ;               _SubAddress_            Sub-Address of slave
```

© 1997 Microchip Technology Inc.

```
00520 ;               _BYTES2_                2st block #of bytes
00521 ;               _SourcePointer2_        Start Pointer of the 2nd block
00522 ;
00523 ;  Sequence :
00524 ;               S-SlvW-A-SubA-A-D1[0]-A-....-D1[N-1]-A-D2[0]-A-.....A-D2[M-1]-A-P
00525 ;
00526 ;  Note : This MACRO is basically same as calling I2C_WR_SUB twice, but
00527 ;               a STOP bit is not sent (bus is not given up) in between
00528 ;               the two I2C_WR_SUB
00529 ;
00530 ;  Check Txmt_Success flag for any transmission errors
00531 ;
00532 ;*****************************************************************************************************************
00533
00534 I2C_WR_SUB_WR   MACRO   _COUNT1_, _SourcePointer1_, _Sub_Address_, _COUNT2_, _SourcePointer2_
00535
00536                 movlw   (_COUNT1_ + 1)
00537                 movwf   tempCount
00538                 movlw   (_SourcePointer1_ - 1)
00539                 movwf   FSR
00540
00541                 movf    INDF,W
00542                 movwf   StoreTemp_1     ; temporarily store contents of (_SourcePointer_ -1)
00543                 movlw   _Sub_Address_
00544                 movwf   INDF            ; store temporarily the sub-address at (_SourcePointer_ -1)
00545                 call    _i2c_block_write        ; write _BYTES_+1 block of data
00546 ;
00547                 movf    StoreTemp_1,W
00548                 movwf   (_SourcePointer1_ - 1)   ; restore contents of (_SourcePointer_ - 1)
00549 ; Block 1 write over
00550 ; Send Block 2
00551                 movlw   _COUNT2_
00552                 movwf   tempCount
00553                 movlw   _SourcePointer2_
00554                 movwf   FSR
00555                 call    _block_wr1_loop
00556 ;
00557                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
00558
00559                 ENDM
00560
00561
00562 ;*****************************************************************************************************************
00563 ;
00564 ;                               I2C_WR_SUB_RD
00565 ;
00566 ;  This macro writes a block of data from SourcePointer of length _COUNT1_ to a slave
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
00567 ;       at sub-address and then Reads a block of Data of length _COUNT2_ to destination
00568 ;       address pointer
00569 ;
00570 ;
00571 ;  Message Structure :
00572 ;              S-SlvW-A-SubA-A-D1[0]-A-.....-A-D1[N-1]-A-S-SlvR-A-D2[0]-A-.....A-D2[M-1]-N-P
00573 ;
00574 ;  Parameters :
00575 ;               _COUNT1_                Length Of Source Buffer
00576 ;               _SourcePointer_         Source Pointer Address
00577 ;               _Sub_Address_           The Sub Address Of the slave
00578 ;               _COUNT2_                The length of Destination Buffer
00579 ;               _DestPointer_           The start address of Destination Pointer
00580 ;
00581 ;*************************************************************************************************************
00582
00583 I2C_WR_SUB_RD   MACRO   _COUNT1_, _SourcePointer_, _Sub_Address_, _COUNT2_, _DestPointer_
00584
00585                 movlw   (_COUNT1_ + 1)
00586                 movwf   tempCount
00587                 movlw   (_SourcePointer_ - 1)
00588                 movwf   FSR
00589
00590                 movf    INDF,W
00591                 movwf   StoreTemp_1     ; temporarily store contents of (_SourcePointer_ -1)
00592                 movlw   _Sub_Address_
00593                 movwf   INDF            ; store temporarily the sub-address at (_SourcePointer_ -1)
00594                 call    _i2c_block_write        ; write _BYTES_+1 block of data
00595 ;
00596                 movf    StoreTemp_1,W
00597                 movwf   (_SourcePointer1_ - 1)   ; restore contents of (_SourcePointer_ - 1)
00598 ;
00599 ; Without sending a STOP bit, read a block of data by using a REPEATED
00600 ; Start Condition
00601 ;
00602                 I2C_READ        _COUNT2_, _DestPointer_
00603
00604                 ENDM
00605
00606
00607 ;*************************************************************************************************************
00608 ;
00609 ;                               I2C_WR_COM_WR
00610 ;
00611 ;  This Macro write 2 blocks of data buffers to a slave in one message. This way no need to give up
00612 ;  the bus after sending the first block.
00613 ;  For example, this kind of transaction is used in an LCD driver where a
```

```
00614 ;       block of control & addresss info is needed and then
00615 ;       another block of actual data to be displayed is needed.
00616 ;
00617 ;
00618 ;  Message Structure :
00619 ;               S-SlvW-A-D1[0]-A-.....A-D1[N-1]-A-D2[0]-A-.....-A-D2[M-1]-A-P
00620 ;  NOTE : This message is same as calling two I2C_WR Macros, except that
00621 ;       the bus is not given up between the sending of 2 blocks (this is
00622 ;       done by not sending a STOP bit inbetween)
00623 ;
00624 ;  Parameters :
00625 ;               _COUNT1_                Length Of Source Buffer #1
00626 ;               _SourcePointer1_        Source Pointer Address of 1st buffer
00627 ;               _COUNT2_                The length of Destination Buffer
00628 ;               _SourcePointer2_        Source Pointer Address of 2nd Buffer
00629 ;
00630 ;*****************************************************************************************************
00631
00632 I2C_WR_COM_WR   MACRO   _COUNT1_, _SourcePointer1_, _COUNT2_, _SourcePointer2_
00633
00634                movlw   _COUNT1_
00635                movwf   tempCount
00636                movlw   _SourcePointer1_
00637                movwf   FSR
00638                call    _i2c_block_write
00639 ;
00640 ; First block sent, now send 2nd block of data
00641 ;
00642                movlw   _COUNT2_
00643                movwf   tempCount
00644                movlw   _SourcePointer2__
00645                movwf   FSR
00646                call    _block_wr1_loop
00647 ;
00648                call    TxmtStopBit     ; End of Double buffer txmt
00649
00650                ENDM
00651
00652
00653 ;*****************************************************************************************************
00654 ;                               INCLUDE I2C Low Level Routines Here
00655 ;*****************************************************************************************************
00656
00657                         include "i2c_low.inc"
00001 ;*****************************************************************************************************
00002 ;
00003 ;                               Low Level I2C Routines
```


© 1997 Microchip Technology Inc.

```
                00004 ;
                00005 ;       Single Master Transmitter & Single Master Receiver Routines
                00006 ;       These routines can very easily be converted to Multi-Master System
                00007 ;       when PIC16C6X with on chip I2C Slave Hardware, Start & Stop Bit
                00008 ;       detection is available.
                00009 ;
                00010 ;   The generic high level routines are given in I2C_HIGH.ASM
                00011 ;
                00012 ;
                00013 ;       Program:          I2C_LOW.ASM
                00014 ;       Revision Date:
                00015 ;                         1-16-97      Compatibility with MPASMWIN 1.40
                00016 ;
                00017 ;*******************************************************************************************************
                00018
                00019
                00020
                00021
                00022
                00023 ;*******************************************************************************************************
                00024 ;                                 I2C Bus Initialization
                00025 ;
                00026 ;*******************************************************************************************************
004D            00027 InitI2CBus_Master:
                00028
004D 1283       00029         bcf     STATUS,RP0
004E 0806       00030         movf    PORTB,W
004F 39FC       00031         andlw   0xFC            ; do not use BSF & BCF on Port Pins
0050 0086       00032         movwf   PORTB           ; set SDA & SCL to zero. From Now on, simply play with tris
                00033         RELEASE_BUS
0051 1683           M         bsf     STATUS,RP0      ; select Bank1
0052 1486           M         bsf     _SDA            ; tristate SDA
0053 1406           M         bsf     _SCL            ; tristate SCL
                    M ;       bcf     _Bus_Busy       ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
0054 0190       00034         clrf    Bus_Status      ; reset status reg
0055 0191       00035         clrf    Bus_Control     ; clear the Bus_Control Reg, reset to 8 bit addressing
0056 0008       00036         return
                00037 ;
                00038 ;*******************************************************************************************************
                00039 ;                                     Send Start Bit
                00040 ;
                00041 ;*******************************************************************************************************
                00042
0057            00043 TxmtStartBit:
0057 1683       00044          bsf     STATUS,RP0      ; select Bank1
0058 1486       00045          bsf     _SDA            ; set SDA high
0059 1406       00046          bsf     _SCL            ; clock is high
```

© 1997 Microchip Technology Inc.

```
                        00047 ;
                        00048 ; Setup time for a REPEATED START condition (4.7 uS)
                        00049 ;
005A 210D               00050                 call    Delay40uSec     ; only necessary for setup time
                        00051 ;
005B 1086               00052                 bcf     _SDA            ; give a falling edge on SDA while clock is high
                        00053 ;
005C 210B               00054                 call    Delay47uSec     ; only necessary for START HOLD time
                        00055 ;
005D 1410               00056                 bsf     _Bus_Busy       ; on a start condition bus is busy
                        00057 ;
005E 0008               00058                 return
                        00059
                        00060
                        00061 ;*****************************************************************************************************
                        00062 ;                                       Send Stop Bit
                        00063 ;
                        00064 ;*****************************************************************************************************
                        00065
005F                    00066 TxmtStopBit:
005F 1683               00067                 bsf     STATUS,RP0      ; select Bank1
0060 1006               00068                 bcf     _SCL
0061 1086               00069                 bcf     _SDA            ; set SDA low
0062 1406               00070                 bsf     _SCL            ; Clock is pulled up
0063 210D               00071                 call    Delay40uSec     ; Setup Time For STOP Condition
0064 1486               00072                 bsf     _SDA            ; give a rising edge on SDA while CLOCK is high
                        00073 ;
                        00074   if _ENABLE_BUS_FREE_TIME
                        00075 ; delay to make sure a START bit is not sent immediately after a STOP, ensure BUS Free Time tBUF
                        00076 ;
0065 210B               00077                 call    Delay47uSec
                        00078   endif
                        00079 ;
0066 1010               00080                 bcf     _Bus_Busy       ; on a stop condition bus is considered Free
                        00081 ;
0067 0008               00082                 return
                        00083
                        00084
                        00085 ;*****************************************************************************************************
                        00086 ;                                       Abort Transmission
                        00087 ;
                        00088 ;   Send STOP Bit & set Abort Flag
                        00089 ;*****************************************************************************************************
                        00090
0068                    00091 AbortTransmission:
                        00092
0068 205F               00093                 call    TxmtStopBit
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
0069 1490          00094                bsf     _Abort
006A 0008          00095                return
                   00096
                   00097 ;*****************************************************************************************************
                   00098 ;                                Transmit Address (1st Byte)& Put in Read/Write Operation
                   00099 ;
                   00100 ;  Transmits Slave Addr On the 1st byte and set LSB to R/W operation
                   00101 ;  Slave Address must be loaded into SlaveAddr reg
                   00102 ;  The R/W operation must be set in Bus_Status Reg (bit _SLAVE_RW): 0 for Write & 1 for Read
                   00103 ;
                   00104 ;  On Success, return TRUE in W register, else FALSE in W register
                   00105 ;
                   00106 ;   If desired, the failure may tested by the bits in Bus Status Reg
                   00107 ;
                   00108 ;*****************************************************************************************************
                   00109
006B               00110 Txmt_Slave_Addr:
006B 1390          00111         bcf     _ACK_Error              ; reset Acknowledge error bit
006C 1C11          00112         btfss   _10BitAddr
006D 2886          00113         goto    SevenBitAddr
                   00114 ;
006E 1C91          00115         btfss   _Slave_RW
006F 287D          00116         goto    TenBitAddrWR            ; For 10 Bit WR simply send 10 bit addr
                   00117 ;
                   00118 ; Required to READ a 10 bit slave, so first send 10 Bit for WR & Then Repeated Start
                   00119 ; and then Hi Byte Only for read opreation
                   00120 ;
0070               00121 TenBitAddrRd:
                   00122
0070 1091          00123         bcf     _Slave_RW               ; temporarily set for WR operation
0071 207D          00124         call    TenBitAddrWR
0072 1E10          00125         btfss   _Txmt_Success           ; skip if successful
0073 3400          00126         retlw   FALSE
                   00127
0074 2057          00128         call    TxmtStartBit            ; send A REPEATED START condition
0075 1491          00129         bsf     _Slave_RW               ; For 10 bit slave Read
                   00130
0076 080D          00131         movf    SlaveAddr+1,W
0077 008E          00132         movwf   DataByte
0078 140E          00133         bsf     DataByte,LSB            ; Read Operation
0079 2095          00134         call    SendData                ; send ONLY high byte of 10 bit addr slave
007A 288C          00135         goto    _AddrSendTest           ; 10 Bit Addr Send For Slave Read Over
                   00136 ;
                   00137 ; if successfully transmitted, expect an ACK bit
                   00138 ;
007B 1E10          00139         btfss   _Txmt_Success           ; if not successful, generate STOP & abort transfer
007C 2890          00140         goto    _AddrSendFail
```

```
                        00141 ;
                        00142
007D                    00143 TenBitAddrWR:
                        00144
007D 080D               00145         movf    SlaveAddr+1,W
007E 008E               00146         movwf   DataByte
007F 100E               00147         bcf     DataByte,LSB            ; WR Operation
                        00148 ;
                        00149 ; Ready to transmit data : If Interrupt Driven (i.e if Clock Stretched LOW Enabled)
                        00150 ; then save RETURN Address Pointer
                        00151 ;
0080 2095               00152         call    SendData                ; send high byte of 10 bit addr slave
                        00153 ;
                        00154 ; if successfully transmitted, expect an ACK bit
                        00155 ;
0081 1E10               00156         btfss   _Txmt_Success           ; if not successful, generate STOP & abort transfer
0082 2890               00157         goto    _AddrSendFail
                        00158 ;
0083 080C               00159         movf    SlaveAddr,W
0084 008E               00160         movwf   DataByte                ; load addr to DatByte for transmission
0085 288B               00161         goto    EndTxmtAddr
                        00162
0086                    00163 SevenBitAddr:
0086 080C               00164         movf    SlaveAddr,W
0087 008E               00165         movwf   DataByte                ; load addr to DatByte for transmission
0088 100E               00166         bcf     DataByte,LSB
0089 1891               00167         btfsc   _Slave_RW               ; if skip then write operation
008A 140E               00168         bsf     DataByte,LSB            ; Read Operation
                        00169
008B                    00170 EndTxmtAddr:
008B 2095               00171         call    SendData                ; send 8 bits of address, bus is our's
                        00172 ;
                        00173 ; if successfully transmitted, expect an ACK bit
                        00174 ;
008C                    00175 _AddrSendTest:
008C 1E10               00176         btfss   _Txmt_Success           ; skip if successful
008D 2890               00177         goto    _AddrSendFail
008E 0064               00178         clrwdt
008F 3401               00179         retlw   TRUE
                        00180 ;
0090                    00181 _AddrSendFail:
0090 0064               00182         clrwdt
0091 1F90               00183         btfss   _ACK_Error
0092 3400               00184         retlw   FALSE                   ; Addr Txmt Unsuccessful, so return 0
                        00185 ;
                        00186 ; Address Not Acknowledged, so send STOP bit
                        00187 ;
```


© 1997 Microchip Technology Inc.

```
0093 205F          00188         call    TxmtStopBit
0094 3400          00189         retlw   FALSE                   ; Addr Txmt Unsuccessful, so return 0
                   00190 ;
                   00191 ;*****************************************************************************************************
                   00192 ;                                Transmit A Byte Of Data
                   00193 ;
                   00194 ; The data to be transmitted must be loaded into DataByte Reg
                   00195 ; Clock stretching is allowed by slave. If the slave pulls the clock low, then, the stretch is detected
                   00196 ; and INT Interrupt on Rising edge is enabled and also TMR0 timeout interrupt is enabled
                   00197 ;       The clock stretching slows down the transmit rate because all checking is done in
                   00198 ;       software. However, if the system has fast slaves and needs no clock stretching, then
                   00199 ;       this feature can be disabled during Assembly time by setting
                   00200 ;       _CLOCK_STRETCH_ENABLED must be set to FALSE.
                   00201 ;
                   00202 ;*****************************************************************************************************
0095               00203 SendData:
                   00204
                   00205 ;
                   00206 ; TXmtByte & Send Data are same, Can check errors here before calling TxmtByte
                   00207 ; For future compatibility, the user MUST call SendData & NOT TxmtByte
                   00208 ;
0095 2896          00209         goto    TxmtByte
                   00210
                   00211 ;
0096               00212 TxmtByte:
0096 080E          00213                 movf    DataByte,W
0097 0093          00214                 movwf   DataByteCopy    ; make copy of DataByte
0098 1510          00215                 bsf     _Txmt_Progress  ; set Bus status for txmt progress
0099 1210          00216                 bcf     _Txmt_Success   ; reset status bit
009A 3008          00217                 movlw   0x08
009B 008F          00218                 movwf   BitCount
009C 1683          00219                 bsf     STATUS,RP0
                   00220     if _CLOCK_STRETCH_CHECK
                   00221 ;         set TMR0 to INT CLK timeout for 1 mSec
                   00222 ;         do not disturb user's selection of RPUB in OPTION Register
                   00223 ;
009D 0801          00224                 movf    OPTION_REG,W
009E 39C3          00225                 andlw   _OPTION_INIT    ; defined in I2C.H header file
009F 0081          00226                 movwf   OPTION_REG
                   00227     endif
                   00228
00A0               00229 TxmtNextBit:
00A0 0064          00230                 clrwdt                  ; clear WDT, set for 18 mSec
00A1 1006          00231                 bcf     _SCL
00A2 0D93          00232                 rlf     DataByteCopy, F ; MSB first, Note DataByte Is Lost
00A3 1086          00233                 bcf     _SDA
00A4 1803          00234                 btfsc   STATUS,C
```

© 1997 Microchip Technology Inc.

```
00A5 1486          00235                 bsf     _SDA
00A6 210B          00236                 call    Delay47uSec     ; guarantee min LOW TIME tLOW & Setup time
00A7 1406          00237                 bsf     _SCL            ; set clock high, check if clock is high, else clock being stretched
00A8 210D          00238                 call    Delay40uSec     ; guarantee min HIGH TIME tHIGH
                   00239         if _CLOCK_STRETCH_CHECK
00A9 1283          00240                 bcf     STATUS,RP0
00AA 0181          00241                 clrf    TMR0            ; clear TMR0
00AB 110B          00242                 bcf     INTCON,T0IF     ; clear any pending flags
00AC 168B          00243                 bsf     INTCON,T0IE     ; elable TMR0 Interrupt
00AD 1391          00244                 bcf     _TIME_OUT_      ; reset timeout error flag
00AE               00245 Check_SCL_1:
00AE 1B91          00246                 btfsc   _TIME_OUT_      ; if TMR0 timeout or Error then Abort & return
00AF 28FC          00247                 goto    Bus_Fatal_Error ; Possible FATAL Error on Bus
00B0 1283          00248                 bcf     STATUS,RP0
00B1 1C06          00249                 btfss   _SCL            ; if clock not being stretched, it must be high
00B2 28AE          00250                 goto    Check_SCL_1     ; loop until SCL high or TMR0 timeout interrupt
00B3 128B          00251                 bcf     INTCON,T0IE     ; Clock good, disable TMR0 interrupts
00B4 1683          00252                 bsf     STATUS,RP0
                   00253         endif
00B5 0B8F          00254                 decfsz  BitCount, F
00B6 28A0          00255                 goto    TxmtNextBit
                   00256 ;
                   00257 ; Check For Acknowledge
                   00258 ;
00B7 1006          00259                 bcf     _SCL            ; reset clock
00B8 1486          00260                 bsf     _SDA            ; Release SDA line for Slave to pull down
00B9 210B          00261                 call    Delay47uSec     ; guarantee min LOW TIME tLOW & Setup time
00BA 1406          00262                 bsf     _SCL            ; clock for slave to ACK
00BB 210D          00263                 call    Delay40uSec     ; guarantee min HIGH TIME tHIGH
00BC 1283          00264                 bcf     STATUS,RP0      ; select Bank0 to test PortB pin SDA
00BD 1886          00265                 btfsc   _SDA            ; SDA should be pulled low by slave if OK
00BE 28C5          00266                 goto    _TxmtErrorAck
                   00267 ;
00BF 1683          00268                 bsf     STATUS,RP0
00C0 1006          00269                 bcf     _SCL            ; reset clock
                   00270
00C1 1110          00271                 bcf     _Txmt_Progress  ; reset TXMT bit in Bus Status
00C2 1610          00272                 bsf     _Txmt_Success   ; transmission successful
00C3 1390          00273                 bcf     _ACK_Error      ; ACK OK
00C4 0008          00274                 return
00C5               00275 _TxmtErrorAck:
                   00276                 RELEASE_BUS
00C5 1683              M                 bsf     STATUS,RP0      ; select Bank1
00C6 1486              M                 bsf     _SDA            ; tristate SDA
00C7 1406              M                 bsf     _SCL            ; tristate SCL
                       M ;               bcf     _Bus_Busy       ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
00C8 1110          00277                 bcf     _Txmt_Progress  ; reset TXMT bit in Bus Status
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
00C9 1210          00278                 bcf     _Txmt_Success   ; transmission NOT successful
00CA 1790          00279                 bsf     _ACK_Error      ; No ACK From Slave
00CB 0008          00280                 return
                   00281 ;
                   00282 ;*****************************************************************************************************
                   00283 ;
                   00284 ;                               Receive  A Byte Of Data From Slave
                   00285 ;
                   00286 ;  assume address is already sent
                   00287 ;  if last byte to be received, do not acknowledge slave (last byte is testted from
                   00288 ;  _Last_Byte_Rcv bit of control reg)
                   00289 ;  Data Received on successful reception is in DataReg register
                   00290 ;
                   00291 ;
                   00292 ;*****************************************************************************************************
                   00293 ;
                   00294
00CC               00295 GetData:
00CC 28CD          00296                 goto    RcvByte
                   00297 ;
00CD               00298 RcvByte:
                   00299
00CD 1590          00300                 bsf     _Rcv_Progress   ; set Bus status for txmt progress
00CE 1290          00301                 bcf     _Rcv_Success    ; reset status bit
                   00302
00CF 3008          00303                 movlw   0x08
00D0 008F          00304                 movwf   BitCount
                   00305         if _CLOCK_STRETCH_CHECK
00D1 1683          00306                 bsf     STATUS,RP0
                   00307 ;        set TMR0 to INT CLK timeout for 1 mSec
                   00308 ;        do not disturb user's selection of RPBU in OPTION Register
                   00309 ;
00D2 0801          00310                 movf    OPTION_REG,W
00D3 39C3          00311                 andlw   _OPTION_INIT    ; defined in I2C.H header file
00D4 0081          00312                 movwf   OPTION_REG
                   00313         endif
                   00314
00D5               00315 RcvNextBit:
00D5 0064          00316                 clrwdt                  ; clear WDT, set for 18 mSec
00D6 1683          00317                 bsf     STATUS,RP0      ; Bank1 for TRIS manipulation
00D7 1006          00318                 bcf     _SCL
00D8 1486          00319                 bsf     _SDA            ; can be removed from loop
00D9 210B          00320                 call    Delay47uSec     ; guarantee min LOW TIME tLOW & Setup time
00DA 1406          00321                 bsf     _SCL            ; clock high, data sent by slave
00DB 210D          00322                 call    Delay40uSec     ; guarantee min HIGH TIME tHIGH
                   00323         if _CLOCK_STRETCH_CHECK
00DC 1283          00324                 bcf     STATUS,RP0
```

```
00DD 0181          00325                 clrf    TMR0            ; clear TMR0
00DE 110B          00326                 bcf     INTCON,T0IF     ; clear any pending flags
00DF 168B          00327                 bsf     INTCON,T0IE     ; enable TMR0 Interrupt
00E0 1391          00328                 bcf     _TIME_OUT_      ; reset timeout error flag
00E1               00329 Check_SCL_2:
00E1 1B91          00330                 btfsc   _TIME_OUT_      ; if TMR0 timeout or Error then Abort & return
00E2 28FC          00331                 goto    Bus_Fatal_Error ; Possible FATAL Error on Bus
00E3 1283          00332                 bcf     STATUS,RP0
00E4 1C06          00333                 btfss   _SCL            ; if clock not being stretched, it must be high
00E5 28E1          00334                 goto    Check_SCL_2     ; loop until SCL high or TMR0 timeout interrupt
00E6 128B          00335                 bcf     INTCON,T0IE     ; Clock good, diable TMR0 interrupts
00E7 1683          00336                 bsf     STATUS,RP0
                   00337         endif
00E8 1283          00338                 bcf     STATUS,RP0      ; select Bank0 to read Ports
00E9 1003          00339                 bcf     STATUS,C
00EA 1886          00340                 btfsc   _SDA
00EB 1403          00341                 bsf     STATUS,C
                   00342 ;                                       ; TEMP ???? DO 2 out of 3 Majority detect
00EC 0D8E          00343                 rlf     DataByte, F     ; left shift data ( MSB first)
00ED 0B8F          00344                 decfsz  BitCount, F
00EE 28D5          00345                 goto    RcvNextBit
                   00346 ;
                   00347 ; Generate ACK bit if not last byte to be read,
                   00348 ; if last byte Generate NACK ; do not send ACK on last byte, main routine will send a STOP bit
                   00349 ;
00EF 1683          00350                 bsf     STATUS,RP0
00F0 1006          00351                 bcf     _SCL
00F1 1086          00352                 bcf     _SDA            ; ACK by pulling SDA low
00F2 1911          00353                 btfsc   _Last_Byte_Rcv
00F3 1486          00354                 bsf     _SDA            ; if last byte, send NACK by setting SDA high
00F4 210B          00355                 call    Delay47uSec     ; guarantee min LOW TIME tLOW & Setup time
00F5 1406          00356                 bsf     _SCL
00F6 210D          00357                 call    Delay40uSec     ; guarantee min HIGH TIME tHIGH
00F7               00358 RcvEnd:
00F7 1006          00359                 bcf     _SCL            ; reset clock
                   00360
00F8 1190          00361                 bcf     _Rcv_Progress   ; reset TXMT bit in Bus Status
00F9 1690          00362                 bsf     _Rcv_Success    ; transmission successful
00FA 1390          00363                 bcf     _ACK_Error      ; ACK OK
                   00364
00FB 0008          00365                 return
                   00366
                   00367   if _CLOCK_STRETCH_CHECK
                   00368 ;*****************************************************************************************************************
                   00369 ;                                 Fatal Error On I2C Bus
                   00370 ;
                   00371 ;  Slave pulling clock for too long or if SCL Line is stuck low.
```


© 1997 Microchip Technology Inc.

```
                00372 ;  This occurs if during Transmission, SCL is stuck low for period longer than appox 1mS
                00373 ;   and TMR0 times out ( appox 4096 cycles : 256 * 16 -- prescaler of 16).
                00374 ;
                00375 ;*****************************************************************************************************
                00376
00FC            00377 Bus_Fatal_Error:
                00378
                00379 ; disable TMR0 Interrupt
                00380 ;
00FC 128B       00381         bcf     INTCON,T0IE                     ; disable TMR0 interrupts, until next TXMT try
                00382
                00383         RELEASE_BUS
00FD 1683           M                         bsf     STATUS,RP0      ; select Bank1
00FE 1486           M                         bsf     _SDA            ; tristate SDA
00FF 1406           M                         bsf     _SCL            ; tristate SCL
                    M ;                       bcf     _Bus_Busy       ; Bus Not Busy, TEMP ????, set/clear on Start & Stop
                00384 ;
                00385 ;  Set the Bus_Status Bits appropriately
                00386 ;
0100 1490       00387         bsf     _Abort                          ; transmission was aborted
0101 1710       00388         bsf     _Fatal_Error                    ; FATAL Error occured
0102 1110       00389         bcf     _Txmt_Progress                  ; Transmission Is Not in Progress
0103 1210       00390         bcf     _Txmt_Success                   ; Transmission Unsuccesful
                00391 ;
0104 205F       00392         call    TxmtStopBit                     ; Try sending a STOP bit, may be not successful
                00393 ;
0105 0008       00394         return
                00395 ;
                00396 ;*****************************************************************************************************
                00397   endif
                00398
                00399
                00400 ;*****************************************************************************************************
                00401 ;                       General Purpose Delay Routines
                00402 ;
                00403 ;  Delay4uS     is wait loop for 4.0 uSec
                00404 ;  Delay47uS    is wait loop for 4.7 uSec
                00405 ;  Delay50uS    is wait loop for 5.0 uSec
                00406 ;
                00407 ;*****************************************************************************************************
                00408 ;
                00409
0106            00410 Delay50uSec:
0106 3006       00411         movlw   ((_50uS_Delay-5)/3 + 1)
0107            00412 DlyK
0107 0092       00413         movwf   DelayCount
0108 0B92       00414         decfsz  DelayCount, F
```

© 1997 Microchip Technology Inc.

```
0109 2908          00415         goto    $-1
010A 0008          00416         return
                   00417 ;
010B               00418 Delay47uSec:
010B 3004          00419         movlw   ((_47uS_Delay-8)/3 + 1)
010C 2907          00420         goto    DlyK
                   00421 ;
010D               00422 Delay40uSec:
010D 3003          00423         movlw   ((_40uS_Delay-8)/3 + 1)
010E 2907          00424         goto    DlyK
                   00425 ;
                   00426 ;*****************************************************************************************************
                   00109   endif
                   00110
                   00111 ;*****************************************************************************************************
                   00112 ;
                   00113
010F               00114 ReadSlave1:
                   00115
                   00116 ;
                   00117 ;  EEPROM (24C04) may be in write mode (busy), check for ACK by sending a control byte
                   00118 ;
                   00119                 LOAD_ADDR_8  _Slave_1_Addr
                       M
010F 1011              M                 bcf     _10BitAddr      ; Set for 8 Bit Address Mode
0110 30A0              M                 movlw   (0xA0 & 0xff)
0111 008C              M                 movwf   SlaveAddr
                       M
0112               00120 wait1:
                   00121                 I2C_TEST_DEVICE
                       M
0112 2011              M                 call    IsSlaveActive   ; TEMP ???? : Assembler Error with this MACRO
                       M
0113 1F11          00122                 btfss   _SlaveActive    ; See If slave is responding
0114 2912          00123                 goto    wait1           ; if stuck forever, recover from WDT, can use other schemes
0115 0064          00124                 clrwdt
                   00125                 I2C_READ_SUB  8, DataBegin+1, 0x50
                       M
0116 1091              M                 bcf     _Slave_RW       ; set for write operation
0117 2057              M                 call    TxmtStartBit    ; send START bit
0118 206B              M                 call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
                       M
0119 3050              M                 movlw   0x50
011A 008E              M                 movwf   DataByte        ; START address of EEPROM(slave 1)
011B 2095              M                 call    SendData        ; write sub address
                       M ;
                       M ; do not send STOP after this, use REPEATED START condition
```


© 1997 Microchip Technology Inc.

```
                    M ;
                    M
                    M                   I2C_READ 8, DataBegin+1
                    M
                    M
011C 3007           M                   movlw   (8 -1)
011D 0096           M                   movwf   tempCount               ; -1 because, the last byte is used out of loop
011E 3021           M                   movlw   DataBegin+1
011F 0084           M                   movwf   FSR                     ; FIFO destination address pointer
                    M
0120 2039           M                   call    _i2c_block_read
                    M
                    M
                    M
                  00126 ;
                  00127 ; Read 8 bytes of data from Slave 2 starting from Sub-Address 0x60
                  00128 ;
                  00129                 LOAD_ADDR_8  _Slave_2_Addr
                    M
0121 1011           M                   bcf     _10BitAddr      ; Set for 8 Bit Address Mode
0122 30AC           M                   movlw   (0xAC & 0xff)
0123 008C           M                   movwf   SlaveAddr
                    M
0124              00130 wait2:
                  00131                 I2C_TEST_DEVICE
                    M
0124 2011           M                   call    IsSlaveActive   ; TEMP ???? : Assembler Error with this MACRO
                    M
0125 1F11         00132                 btfss   _SlaveActive    ; See If slave is responding
0126 2924         00133                 goto    wait2           ; if stuck forever, recover from WDT, can use other schemes
0127 0064         00134                 clrwdt
                  00135                 I2C_READ_SUB  8, DataBegin+1, 0x60
                    M
0128 1091           M                   bcf     _Slave_RW       ; set for write operation
0129 2057           M                   call    TxmtStartBit    ; send START bit
012A 206B           M                   call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
                    M
012B 3060           M                   movlw   0x60
012C 008E           M                   movwf   DataByte        ; START address of EEPROM(slave 1)
012D 2095           M                   call    SendData        ; write sub address
                    M ;
                    M ; do not send STOP after this, use REPEATED START condition
                    M ;
                    M
                    M                   I2C_READ 8,       DataBegin+1
                    M
                    M
```

© 1997 Microchip Technology Inc.

```
012E 3007              M                movlw   (8 -1)
012F 0096              M                movwf   tempCount               ; -1 because, the last byte is used out of loop
0130 3021              M                movlw   DataBegin+1
0131 0084              M                movwf   FSR                     ; FIFO destination address pointer
                       M
0132 2039              M                call    _i2c_block_read
                       M
                       M
                       M
                    00136
                    00137
0133 0008           00138        return
                    00139 ;
                    00140 ;*****************************************************************************************************
                    00141
0134                00142 ReadSlave3:
                    00143
                    00144                LOAD_ADDR_8  _Slave_3_Addr
                       M
0134 1011              M                bcf     _10BitAddr      ; Set for 8 Bit Address Mode
0135 30D6              M                movlw   (0xD6 & 0xff)
0136 008C              M                movwf   SlaveAddr
                       M
0137                00145 wait3:
                    00146                I2C_TEST_DEVICE
                       M
0137 2011              M                call    IsSlaveActive   ; TEMP ???? : Assembler Error with this MACRO
                       M
0138 1F11           00147                btfss   _SlaveActive    ; See If slave is responding
0139 2937           00148                goto    wait3           ; if stuck forever, recover from WDT, can use other schemes
013A 0064           00149                clrwdt
                    00150                I2C_READ_SUB  8, DataBegin, 0
                       M
013B 1091              M                bcf     _Slave_RW       ; set for write operation
013C 2057              M                call    TxmtStartBit    ; send START bit
013D 206B              M                call    Txmt_Slave_Addr ; if successful, then _Txmt_Success bit is set
                       M
013E 3000              M                movlw   0
013F 008E              M                movwf   DataByte        ; START address of EEPROM(slave 1)
0140 2095              M                call    SendData        ; write sub address
                       M ;
                       M ; do not send STOP after this, use REPEATED START condition
                       M ;
                       M
                       M                I2C_READ 8, DataBegin
                       M
                       M
```


© 1997 Microchip Technology Inc.

```
0141 3007            M                movlw   (8 -1)
0142 0096            M                movwf   tempCount              ; -1 because, the last byte is used out of loop
0143 3020            M                movlw   DataBegin
0144 0084            M                movwf   FSR                    ; FIFO destination address pointer
                     M
0145 2039            M                call    _i2c_block_read
                     M
                     M
                     M
                   00151 ;
                   00152
0146 0008          00153         return
                   00154
                   00155 ;*****************************************************************************************************
                   00156 ;
                   00157 ;                        Fill Data Buffer With Test Data ( 8 bytes of 0x55, 0xAA pattern)
                   00158 ;
                   00159 ;*****************************************************************************************************
                   00160
0147               00161 FillDataBuf:
                   00162
0147 3000          00163         movlw   0x00             ; start address location of EEPROM array
0148 00A0          00164         movwf   DataBegin        ; 1st byte of data to be sent is start address
0149 3021          00165         movlw   DataBegin+1      ; data starts following address (RAM Pointer)
014A 0084          00166         movwf   FSR
014B 3008          00167         movlw   8                ; fill RAM with 8 bytes , this data is written to EEPROM (slave)
014C 009A          00168         movwf   byteCount
014D 3055          00169         movlw   0x55             ; pattern to fill with is 0x55 & 0xAA
014E 009B          00170         movwf   HoldData
014F               00171 X1:
014F 099B          00172         comf    HoldData, F
0150 081B          00173         movf    HoldData,W
0151 0080          00174         movwf   INDF
0152 0A84          00175         incf    FSR, F           ; point to next location
0153 0B9A          00176         decfsz  byteCount, F
0154 294F          00177         goto    X1
0155 0008          00178         return
                   00179 ;
                   00180 ;*****************************************************************************************************
                   00181 ;
                   00182 ;                           Main Routine  (Test Program)
                   00183 ;
                   00184 ;                        SINGLE  MASTER, MULTIPLE SLAVES
                   00185 ;
                   00186 ;*****************************************************************************************************
                   00187
0156               00188 Start:
```

© 1997 Microchip Technology Inc.

```
0156 204D          00189         call    InitI2CBus_Master       ; initialize I2C Bus
0157 178B          00190         bsf     INTCON,GIE              ; enable global interrupts
                   00191 ;
                   00192
0158 2147          00193         call    FillDataBuf             ; fill data buffer with 8 bytes of data (0x55, 0xAA)
                   00194 ;
                   00195 ; Use high level Macro to send 9 bytes to Slave (1 & 2 : TWO 24C04) of 8 bit Addr
                   00196 ;
                   00197 ;  Write 9 bytes to Slave 1, starting at RAM addr pointer DataBegin
                   00198 ;
                   00199
0159 1810          00200  btfsc  _Bus_Busy ; is Bus Free, ie.has a start & stop bit been detected (only for multi master system)
015A 2959          00201         goto    $-1                     ; a very simple test, unused for now
                   00202
                   00203         LOAD_ADDR_8  _Slave_1_Addr
                       M
015B 1011              M                 bcf     _10BitAddr      ; Set for 8 Bit Address Mode
015C 30A0              M                 movlw   (0xA0 & 0xff)
015D 008C              M                 movwf   SlaveAddr
                       M
                   00204         I2C_WR  0x09, DataBegin
                       M
015E 3009              M                 movlw   0x09
015F 0096              M                 movwf   tempCount
0160 3020              M                 movlw   DataBegin
0161 0084              M                 movwf   FSR
                       M
0162 2019              M                 call    _i2c_block_write
0163 205F              M                 call    TxmtStopBit     ; Issue a stop bit for slave to end transmission
                       M
                   00205 ;
                   00206 ; Write 8 bytes of Data to slave 2 starting at slaves memory address 0x30
                   00207 ;
                   00208
0164 1810          00209  btfsc   _Bus_Busy  ; is Bus Free, ie. has a start & stop bit been detected (only for multi master system)
0165 2964          00210  goto    $-1             ; a very simple test, unused for now
                   00211
                   00212  LOAD_ADDR_8  _Slave_2_Addr
                       M
0166 1011              M                 bcf     _10BitAddr      ; Set for 8 Bit Address Mode
0167 30AC              M                 movlw   (0xAC & 0xff)
0168 008C              M                 movwf   SlaveAddr
                       M
                   00213         I2C_WR_SUB    0x08, DataBegin+1, 0x30
                       M
0169 3009              M                 movlw   (0x08 + 1)
016A 0096              M                 movwf   tempCount
```


© 1997 Microchip Technology Inc.

```
                    M
016B 3020           M               movlw   (DataBegin+1 - 1)
016C 0084           M               movwf   FSR
                    M
016D 0800           M               movf    INDF,W
016E 0097           M               movwf   StoreTemp_1         ; temporarily store contents of (_SourcePointer_ -1)
016F 3030           M               movlw   0x30
0170 0080           M               movwf   INDF                ; store temporarily the sub-address at (_SourcePointer_ -1)
                    M
0171 2019           M               call    _i2c_block_write    ; write _BYTES_+1 block of data
                    M
0172 0817           M               movf    StoreTemp_1,W
0173 00A0           M               movwf   (DataBegin+1 - 1)   ; restore contents of (_SourcePointer_ - 1)
                    M
0174 205F           M               call    TxmtStopBit         ; Issue a stop bit for slave to end transmission
                    M
                  00214
0175 210F         00215             call    ReadSlave1          ; read a byte from slave from current address
                  00216 ;
                  00217     LOAD_ADDR_8  _Slave_3_Addr
                    M
0176 1011           M               bcf     _10BitAddr          ; Set for 8 Bit Address Mode
0177 30D6           M               movlw   (0xD6 & 0xff)
0178 008C           M               movwf   SlaveAddr
                    M
0179 30CC         00218             movlw   0xCC
017A 00A0         00219             movwf   DataBegin
                  00220     I2C_WR_SUB      0x01,DataBegin, 0x33
                    M
017B 3002           M               movlw   (0x01 + 1)
017C 0096           M               movwf   tempCount
                    M
017D 301F           M               movlw   (DataBegin  - 1)
017E 0084           M               movwf   FSR
                    M
017F 0800           M               movf    INDF,W
0180 0097           M               movwf   StoreTemp_1         ; temporarily store contents of (_SourcePointer_ -1)
0181 3033           M               movlw   0x33
0182 0080           M               movwf   INDF                ; store temporarily the sub-address at (_SourcePointer_ -1)
                    M
0183 2019           M               call    _i2c_block_write    ; write _BYTES_+1 block of data
                    M
0184 0817           M               movf    StoreTemp_1,W
0185 009F           M               movwf   (DataBegin - 1)     ; restore contents of (_SourcePointer_ - 1)
                    M
0186 205F           M               call    TxmtStopBit         ; Issue a stop bit for slave to end transmission
                    M
```

© 1997 Microchip Technology Inc.

```
                    00221 ;
0187 2134           00222         call    ReadSlave3              ; Read From Slave PIC
                    00223 ;
                    00224
0188 0064           00225 self    clrwdt
0189 2988           00226         goto    self
                    00227 ;
                    00228 ;*****************************************************************************************************************
                    00229
                    00230         END


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXX------ ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   391
Program Memory Words Free:   633


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     4 suppressed
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN555

Software Implementation of Asynchronous Serial I/O

Author:	*Amar Palacherla* *Microchip Technology*
Code Update:	*Scott Fink* *Microchip Technology Inc.*

INTRODUCTION

PIC16CXXX microcontrollers from Microchip Technology, Inc., high-performance, EPROM-based 8-bit microcontrollers. Some of the members of this series (like the PIC16C71 and PIC16C84) do not have an on-chip hardware asynchronous serial port. This application note describes the interrupt driven software implementation of Asynchronous Serial I/O (Half Duplex RS-232 Communications) using PIC16CXXX microcontrollers. These microcontrollers can operate at very high speeds with a minimum of 250 ns cycle time (with input clock frequency of 16 MHz). To test the RS-232 routines, a simple Digital Voltmeter (DVM)/Analog Data Acquisition System has been implemented using a PIC16C71, in which, upon reception of a command from host (IBM PC-AT®), an 8-bit value of the selected A/D channel is transmitted back to the host.

IMPLEMENTATION

A half duplex, interrupt driven, software implementation of RS-232 communications, using a PIC16C71, is described in detail below. The transmit pin used in the example code is RB7 and the receive pin is connected to the RA4/T0CKI pin (Figure 2). Of course these pins are connected with appropriate voltage translation to/from RS-232/CMOS levels. Schematics describe the voltage translation in the hardware section of this application note.

Transmit Mode

Transmit mode is quite straight-forward to implement in software using interrupts. Once input clock frequency and baud rate are known, the number of clock cycles per bit can be computed. The on-chip Timer0 timer with its prescaler can be used to generate an interrupt on TMR0 overflow. This TMR0 overflow interrupt can be used as timing to send each bit. The Input clock frequency (`_ClkIn`) and Baud Rate (`_BaudRate`) are programmable by the user and the TMR0 time-out value (the period for each bit) is computed at assembly time. Whether the prescaler must be assigned to Timer0 or not is also determined at assembly time. This computation is done in the header file `rs232.h`. Note that very high speed transmissions can be obtained if transmission is done with "software delays" instead of being "every interrupt" driven, however, the processor will be totally dedicated to this job.

Transmission of a byte is performed by calling the `PutChar` function and the data byte in the TxReg is transmitted out. Before calling this function (`PutChar`), the data must be loaded into TxReg and ensure the serial port is free. The serial port is free when both the _txmtProgress and the _rcvOver bits are cleared (see description of these bits in the Serial Status/Control Reg table given later).

Summary of `PutChar` *function:*

1. Make sure _txmtProgress & _rcvOver bits are cleared
2. Load TxReg with data to be transmitted
3. Call `PutChar` function

Receive Mode

The reception mode implementation is slightly different from the transmit mode. Unlike the transmit pin (TX in the example code is RB7, but could be any I/O pin), the receive pin (RX) must be connected to pin RA4/T0CKI. This is because, in reception, the Start Bit, which is asynchronous in nature, must be detected. To detect the Start bit, when put in Reception mode, the Timer0 module is configured to Counter mode. The OPTION register is configured so the Timer0 module is put in Counter mode (increment on external clock on RA4/T0CKI Pin) and set to increment on the falling edge of pin RA4/T0CKI with no prescaler assigned. After this configuration setup, TMR0 (File Reg 1) is loaded with 0xFF. A falling edge on the T0CKI pin makes TMR0 roll over from 0xFF to 0x00, thus generating an interrupt indicating a Start Bit. The RA4/T0CKI pin is sampled again to make sure the transition on TMR0 is not a glitch. Once the start bit has been detected, the Timer0 module is reconfigured to increment on internal clock and the prescaler is assigned to it depending on input master clock frequency and the baud rate (configured same way as the transmission mode).

The software serial port is put in reception mode when a call is made to function `GetChar`. Before calling this function make sure the serial port is free (i.e., _txmtProgress and _rcvOver status bits must be '0'). On completion of a reception of a byte, the data is stored in RxReg and the _rcvOver bit is cleared.

Summary of `GetChar` *function:*

1. Make sure _txmtProgress & _rcvOver bits are cleared.
2. Call `GetChar` function.
3. The received Byte is in TxReg after the _rcvOver bit is cleared.

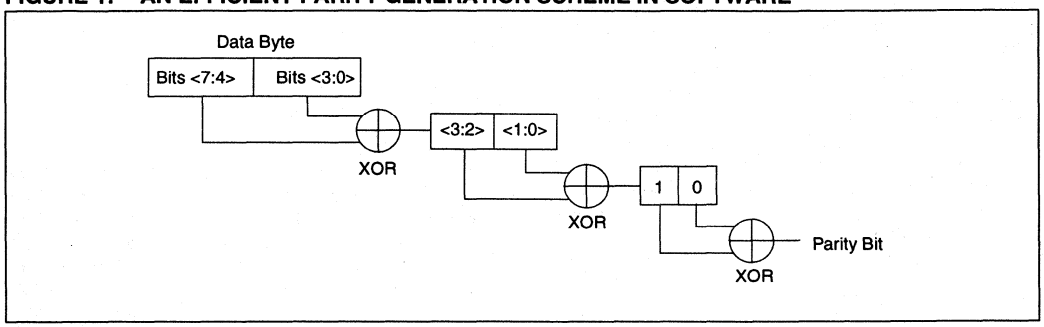
IBM PC-AT is a registered trademark of International Business Machines Corp.

© 1997 Microchip Technology Inc.

Parity Generation

Parity can be enabled at assembly time by setting the "_PARITY_ENABLE" flag to TRUE. If enabled, parity can be configured to either EVEN or ODD parity. In transmission mode, if parity is enabled, the parity bit is computed and transmitted as the ninth bit. On reception, the parity is computed on the received byte and compared to the ninth bit received. If a match does not occur the parity error bit is set in the RS-232 Status/Control Register (_ParityErr bit of SerialStatus reg). The parity bit is computed using the algorithm shown in Figure 1. This algorithm is highly efficient using the PIC16CXXX's `SWAPF` and `XORWF` instructions (with ability to have the destination as either the file register itself or the W register) and the sub-routine (called `GenParity`) is in file `txmtr.asm`.

FIGURE 1: AN EFFICIENT PARITY GENERATION SCHEME IN SOFTWARE



Assembly Time Options

The firmware is written as a general purpose routine and the user must specify the parameters shown in Table 1 before assembling the program. The Status/Control register is described in Table 2.

TABLE 1: LIST OF ASSEMBLY TIME OPTIONS

`_ClkIn`	Input clock frequency of the processor.
`_BaudRate`	Desired Baud Rate. Any valid value can be used. The highest baud rate achievable depends on input clock frequency. 600 to 4800 Baud was tested using a 4 MHz Input Clock. 600 to 19200 Baud was tested using a 10 MHz Input Clock. Higher rates can be obtained using higher input clock frequencies. Once the _BaudRate & _ClkIn are specified, the program automatically selects all the appropriate timings.
`_DataBits`	Can specify 1 to 8 data bits.
`_StopBits`	Limited to 1 Stop Bit. Must be set.
`_PARITY_ENABLE`	Parity Enable Flag. Configure it to TRUE or FALSE. If PARITY is used, then configure it to TRUE, else FALSE. See "_ODD_PARITY" flag description below.
`_ODD_PARITY`	Configure it to TRUE or FALSE. If TRUE, then ODD PARITY is used, else EVEN Parity Scheme is used. This Flag is ignored if _PARITY_ENABLE is configured to FALSE.
`_USE_RTSCTS`	RTS & CTS Hardware handshaking signals. If configured to FALSE, no hardware handshaking is used. If configured to TRUE, RTS & CTS use up 2 I/O Pins of PORTB.

© 1997 Microchip Technology Inc.

TABLE 2: BIT ASSIGNMENT OF SERIAL STATUS/CONTROL REGISTER (SERIALSTATUS REG)

Bit #	Name	Description
0	_txmtProgress	1 = Transmission in progress. 0 = Transmission line free.
1	_txmtEnable	Set this bit on initialization to enable transmission. This bit may be used to abort a transmission. The transmission is aborted if in the middle of a transmission (i.e., when _txmtProgress bit is '1') _txmtEnable bit is cleared. This bit gets automatically set when the PutChar function is called.
2	_rcvProgress	1 = Middle of a byte reception. 0 = Reception of a byte (in RxReg) is complete and is set when a valid start bit is detected in reception mode.
3	_rcvOver	0 = Completion of reception of a byte. The user's code can poll this bit after calling the GetChar function and check to see if it is set. When set, the received byte is in RxReg. Other status bits should also be checked for any reception errors.
4	_ParityErr	1 = Parity error on reception (irrespective of Even Or Odd parity chosen). Not applicable if No Parity is used.
5	_FrameErr	1 = Framing error on reception.
6		Unused
7	_parityBit	The 9th bit of transmission or reception. In transmission mode, the parity bit of the byte to be transmitted is stored in this bit. In receive mode, the 9th bit (or parity bit) received is stored in this bit. Not Applicable if no parity is used.

© 1997 Microchip Technology Inc.

Hardware

The hardware is primarily concerned with voltage translation from RS-232 to CMOS levels and vice versa. Three circuits are given below and the user may choose whichever best applies. The primary difference between each solution is cost versus number of components. Circuits in Figure 3 and Figure 4 are very low cost but have more components than the circuit in Figure 2. The circuit in Figure 2 interfaces to a RS-232 line using a single chip (MAX-232) and single +5V supply. The circuit in Figure 3 is a low cost RS-232 Interface but requires two chips and a single +5V supply source.

Figure 4 shows a very low cost RS-232 Interface to an IBM PC-AT with no external power requirements. The circuit draws power from the RS-232 line (DTR) and meets the spec of drawing power less than 5 mA. This requires that for the host to communicate it must assert lines DTR high and RTS low. The power is drawn from the DTR line and this requires that DTR be asserted high and must be at least 7V. The negative -5 to -10V required by LM339 is drawn from the RTS line and thus the host must assert RTS low. This circuit is possible because of the low current consumption of the PIC16C71 (typical 2 mA).

FIGURE 2: SINGLE CHIP RS-232 INTERFACE (SINGLE +5V SUPPLY)

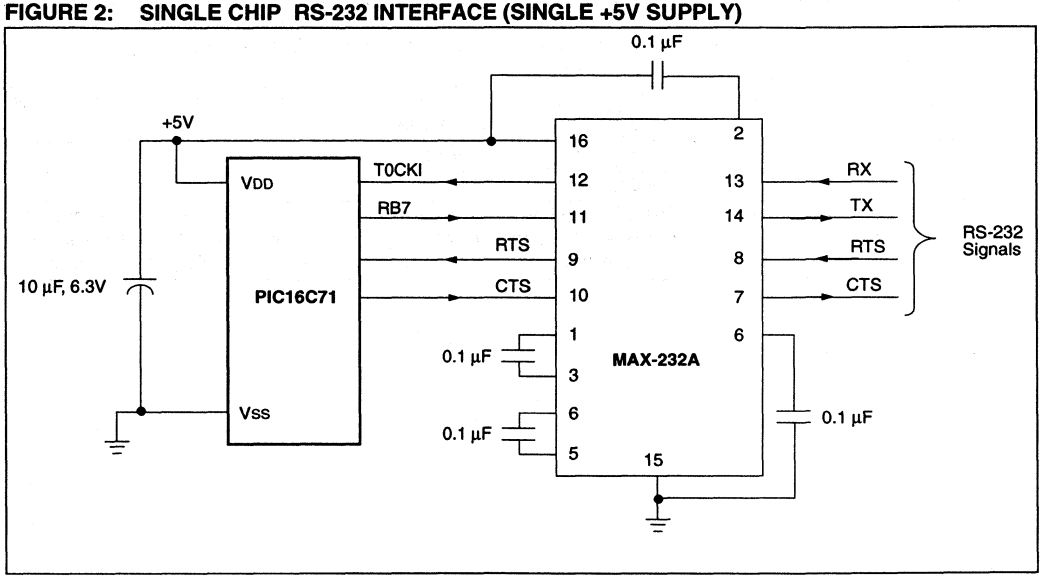
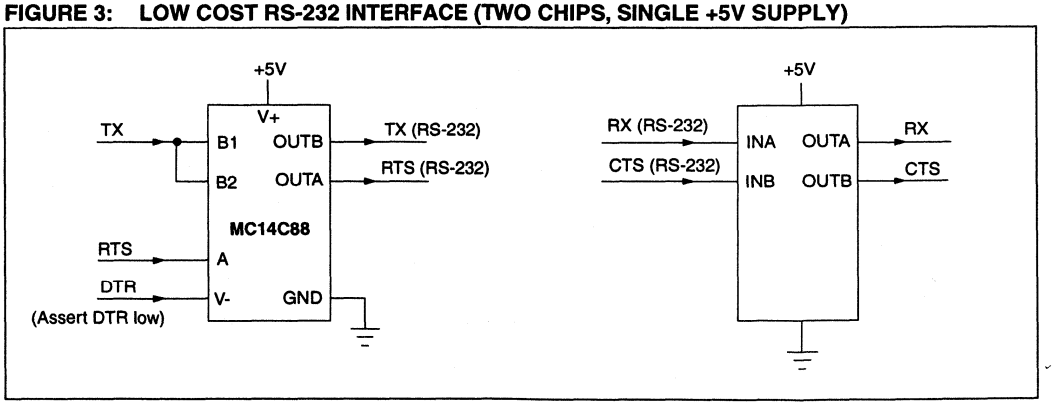
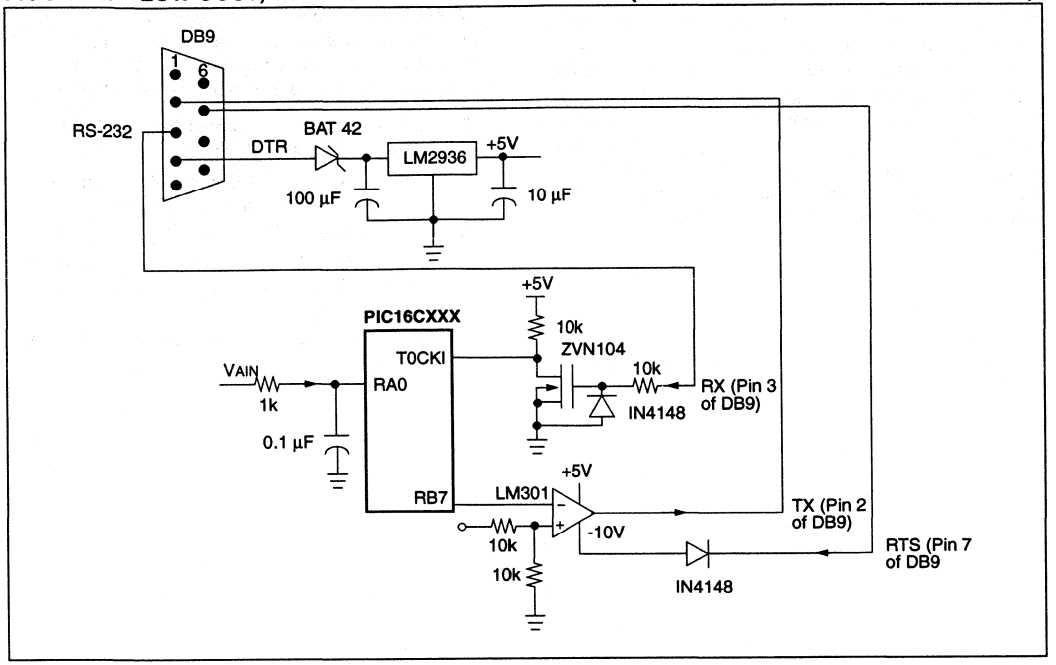


FIGURE 3: LOW COST RS-232 INTERFACE (TWO CHIPS, SINGLE +5V SUPPLY)



© 1997 Microchip Technology Inc.

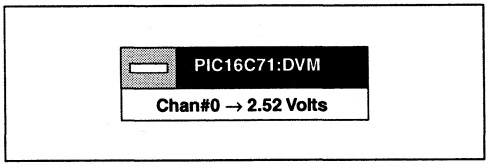
FIGURE 4: LOW COST, LOW POWER RS-232 INTERFACE (POWER SUPPLIED BY RS-232 LINES)



Test Program

To test the transmission and reception modules, a main program is written in which the PIC16C71 waits to receive a command from a host through the RS-232. On reception of a byte (valid commands are 0x00, 0x01, 0x02 & 0x03), the received byte is treated as the PIC16C71's A/D channel number and the requested channel is selected. An A/D conversion is started and when the conversion is complete (in about 20 µs) the digital data (8-bits) is transmitted back to the host. A Microsoft® Windows® program running on an IBM PC/AT was written to act as a host and collect the A/D data from the PIC16C71 via an RS-232 port. The Windows program (DVM.EXE) runs as a background job and displays the A/D data in a small window (similar to the CLOCK program that comes with MS Windows). The windows program and the PIC16C71 together act like a data acquisition system or a digital voltmeter (DVM). The block diagram of the system is shown in Figure 2. The input clock frequency is fixed at 4 MHz and RS-232 parameters are set to 1200 Baud, 8-bits, 1 Stop Bit and No Parity. The program during development stage was also tested at 1200, 2400, 4800 Baud Rates @ 4 MHz Input Clock and up to 19200 Baud @ 10 MHz input clock frequency (all tests were performed with No Parity, Even Parity and Odd Parity at 8 and 7 Data Bits).

FIGURE 5: MS WINDOWS PROGRAM FETCHING A/D DATA FROM PIC16C71 VIA RS-232



Microsoft and Windows are registered trademarks of Microsoft Corp.

Source Code

The PIC16CXXX source code along with the Microsoft Windows DVM Program (executable running on an IBM PC/AT under MS Windows 3.1 or higher) is available on Microchip's BBS. The assembly code for PIC16CXXX must be assembled using Microchip's Universal Assembler, MPASM. The code cannot be assembled using the older assemblers without significant modifications. It is suggested that user's who do not have the new assembler MPASM, change to the new version.

The MS Windows Program (DVM.EXE) runs under MS Windows 3.1 or higher. The program does not have any menus and shows up as a small window displaying A/D Data and runs as a background job. There are a few command line options which are described below.

-Px : x is the comm port number (e.g., - P2 selects COM2). Default is COM1.

-Cy : y is the number of A/D channels to display. Default is one channel (channel #1).

-Sz : z is a floating point number that represents the scaling factor (For example - S5.5 would display the data as 5.5*<8bit A/D>/256). The default value is 5.0 volts.

-S0 : will display the data in raw format without any scaling.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: RS232.H

```
          NOLIST

;*************************************************************************************
;                                    RS-232 Header File
;   PIC16C6X/7X/8X
;*************************************************************************************

_ClkOut           equ     (_ClkIn >> 2)          ; Instruction Cycle Freq = CLKIN/4
;

_CyclesPerBit     set   (_ClkOut/_BaudRate)
_tempCompute      set   (_CyclesPerBit >> 8)
;
;*************************************************************************************
;  Auto Generation Of Prescaler & TMR0 Values
;   Computed during Assembly Time
;*************************************************************************************

;  At first set Default values for TMR0Prescale & TMR0PreLoad
;
TMR0Prescale      set  0
TMR0PreLoad       set  _CyclesPerBit
UsePrescale       set  FALSE

 if (_tempCompute >= 1)
 TMR0Prescale     set  0
 TMR0PreLoad      set  (_CyclesPerBit >> 1)

        UsePrescale     set  TRUE
 endif

 if (_tempCompute >= 2)
 TMR0Prescale     set  1
 TMR0PreLoad      set  (_CyclesPerBit >> 2)
 endif

 if (_tempCompute >= 4)
 TMR0Prescale     set  2
 TMR0PreLoad      set  (_CyclesPerBit >> 3)
 endif

 if (_tempCompute >= 8)
 TMR0Prescale     set  3
 TMR0PreLoad      set  (_CyclesPerBit >> 4)
 endif


 if (_tempCompute >= 16)
 TMR0Prescale     set  4
 TMR0PreLoad      set  (_CyclesPerBit >> 5)
 endif


 if (_tempCompute >= 32)
 TMR0Prescale     set  5
 TMR0PreLoad      set  (_CyclesPerBit >> 6)
 endif


 if (_tempCompute >= 64)
```

```
 TMR0Prescale     set  6
 TMR0PreLoad      set  (_CyclesPerBit >> 7)
 endif


 if   (_tempCompute >= 128)
 TMR0Prescale     set  7
 TMR0PreLoad      set  (_CyclesPerBit >> 8)
 endif

;
     if( (TMR0Prescale == 0) && (TMR0PreLoad < 60))
  messg "Warning : Baud Rate May Be Too High For This Input Clock"
     endif
;
; Compute TMR0 & Prescaler Values For 1.5 Times the Baud Rate for Start Bit Detection
;

_SBitCycles       set  (_ClkOut/_BaudRate) + ((_ClkOut/4)/_BaudRate)
_tempCompute      set  (_SBitCycles >> 8)

_BIT1_INIT        set  08
SBitPrescale      set  0
SBitTMR0Load      set  _SBitCycles


 if (_tempCompute >= 1)
 SBitPrescale     set  0
 SBitTMR0Load     set  (_SBitCycles >> 1)
 _BIT1_INIT       set  0
 endif

 if (_tempCompute >= 2)
 SBitPrescale     set  1
 SBitTMR0Load     set  (_SBitCycles >> 2)
 endif

 if (_tempCompute >= 4)
 SBitPrescale     set  2
 SBitTMR0Load     set  (_SBitCycles >> 3)
 endif

 if (_tempCompute >= 8)
 SBitPrescale     set  3
 SBitTMR0Load     set  (_SBitCycles >> 4)
 endif


 if (_tempCompute >= 16)
 SBitPrescale     set  4
 SBitTMR0Load     set  (_SBitCycles >> 5)
 endif


 if (_tempCompute >= 32)
 SBitPrescale     set  5
 SBitTMR0Load     set  (_SBitCycles >> 6)
 endif


 if (_tempCompute >= 64)
 SBitPrescale     set  6
 SBitTMR0Load     set  (_SBitCycles >> 7)
 endif
```


© 1997 Microchip Technology Inc.

```
 if   (_tempCompute >= 128)
 SBitPrescale     set  7
 SBitTMR0Load     set  (_SBitCycles >> 8)
 endif

;
;*****************************************************************************************
;
#define   _Cycle_Offset1    24       ;account for interrupt latency, call time

LOAD_TMR0    MACRO  Mode, K, Prescale

    if(UsePrescale == 0 && Mode == 0)
 movlw -K + _Cycle_Offset1
    else
 movlw -K + (_Cycle_Offset1 >> (Prescale+1)) ; Re Load TMR0 init value + INT Latency Offset
    endif
 movwf _TMR0      ; Note that Prescaler is cleared when TMR0 is written

  ENDM
;*****************************************************************************************

LOAD_BITCOUNT    MACRO

 movlw _DataBits+1
 movwf BitCount
 movlw 1
 movwf ExtraBitCount

    if  _PARITY_ENABLE
 movlw 2
 movwf ExtraBitCount
    endif

  ENDM
;
;*****************************************************************************************
;   Pin Assignements
;*****************************************************************************************
#define RX_MASK 0x10            ; RX pin is connected to RA4, ie. bit 4
#define RX_Pin      _porta,4    ; RX Pin : RA4
#define RX          RxTemp,4

#define TX          _portb,7    ; TX Pin , RB7

#define _RTS        _portb,5    ; RTS Pin, RB5, Output signal
#define _CTS        _portb,6    ; CTS Pin, RB6, Input signal


#define _txmtProgress      SerialStatus,0
#define _txmtEnable        SerialStatus,1

#define _rcvProgress       SerialStatus,2
#define _rcvOver           SerialStatus,3
#define _ParityErr         SerialStatus,4
#define _FrameErr          SerialStatus,5

#define _parityBit         SerialStatus,7

;*****************************************************************************************

_OPTION_SBIT    set  0x38   ; Increment on Ext Clock (falling edge), for START Bit Detect
  if UsePrescale
_OPTION_INIT    set  0x00   ; Prescaler is used depending on Input Clock & Baud Rate
  else
```


© 1997 Microchip Technology Inc.

```
_OPTION_INIT    set  0x0F
  endif

  CBLOCK  0x0C
          TxReg               ; Transmit Data Holding/Shift Reg
          RxReg               ; Rcv Data Holding Reg
          RxTemp
          SerialStatus        ; Txmt & Rev Status/Control Reg
          BitCount
          ExtraBitCount       ; Parity & Stop Bit Count
          SaveSaveWREG        ; temp hold reg of W register on INT
          SaveStatus          ; temp hold reg of STATUS Reg on INT
          temp1, temp2
  ENDC

;*************************************************************************************************

  LIST
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B

```
TITLE           "RS232 Communications : Half Duplex : PIC16C6x/7x/8x"
SUBTITLE        "Software Implementation : Interrupt Driven"

;*****************************************************************************************
;                       Software Implementation Of RS232 Communications Using PIC16CXXX
;                                               Half-Duplex
;
;  These routines are intended to be used with PIC16C6X/7X family. These routines can be
;  used with processors in the 16C6X/7X family which do not have on board Hardware Async
;  Serial Port.
;  MX..
;
;  Description :
;               Half Duplex RS-232 Mode Is implemented in Software.
;               Both Reception & Transmission are Interrupt driven
;               Only 1 peripheral (TMR0) used for both transmission & reception
;               TMR0 is used for both timing generation (for bit transmission & bit polling)
;               and Start Bit Detection in reception mode.
;               This is explained in more detail in the Interrupt Subroutine.
;               Programmable Baud Rate (speed depending on Input Clock Freq.), programmable
;               #of bits, Parity enable/disable, odd/even parity is implemented.
;               Parity & Framing errors are detected on Reception
;
;                                   RS-232 Parameters
;
;The RS-232 Parameters are defined as shown below:
;
;               _ClkIn          :       Input Clock Frequency of the processor
;                                       NOTE : RC Clock Mode Is Not Suggested due to wide variations)
;               _BaudRate       :       Desired Baud Rate. Any valid value can be used.
;                                       The highest Baud Rate achievable depends on Input Clock Freq.
;                                       300 to 4800 Baud was tested using 4 Mhz Input Clock
;                                       300 to 19200 Baud was tested using 10 Mhz Input Clock
;                                       Higher rates can be obtained using higher Input Clock Frequencies.
;                                       Once the _BaudRate & _ClkIn are specified the program
;                                       automatically selects all the appropriate timings
;               _DataBits       :       Can specify 1 to 8 Bits.
;               _StopBits       :       Limited to 1 Stop Bit. Must set it to 1.
;               _PARITY_ENABLE  :       Parity Enable Flag. Set it to TRUE or FALSE. If PARITY
;                                       is used, then set it to TRUE, else FALSE. See "_ODD_PARITY" flag
;                                       description below
;               _ODD_PARITY     :       Set it to TRUE or FALSE. If TRUE, then ODD PARITY is used, else
;                                       EVEN Parity Scheme is used.
;                                       This Flag is ignored if _PARITY_ENABLE is set to FALSE.
;
;
;  Usage :
;               An example is given in the main program on how to Receive & Transmit Data
;               In the example, the processor waits until a command is received. The command is interpreted
;               as the A/D Channel Number of PIC16C71. Upon reception of a command, the desired A/D channel
;               is selected and after A/D conversion, the 8 Bit A/D data is transmitted back to the Host.
;
;                       The RS-232 Control/Status Reg's bits are explained below :
;
;       "SerialStatus"          : RS-232 Status/Control Register
;
;       Bit 0   :       _txmtProgress   (1 if transmission in progress, 0 if transmission is
;                                       complete) After a byte is transmitted by calling "PutChar" function
;                                       the user's code can poll this bit to check if transmission is
;                                       This bit is reset after the STOP bit has been transmitted complete.
```

```
;       Bit 1   :       _txmtEnable     Set this bit to 1 on initialization to enable transmission.
;                                       This bit can be used to Abort a transmission while the
;                                       transmitter is in progress (i.e when _txmtProgress = 1)
;       Bit 2   :       _rcvProgress    Indicates that the receiver is in middle of reception.
;                                       It is reset when a byte is received.
;       Bit 3   :       _rcvOver        This bit indicates the completion of Reception of a Byte. The
;                                      user's code can poll this bit after calling "GetChar" function.Once
;                                       "GetChar" function is called, this bit is 1 and clear to 0 after
;                                       reception of a complete byte (parity bit if enabled & stop bit)
;       Bit 4   :       _ParityErr      A 1 indicates Parity Error on Reception (both even & odd parity)
;       Bit 5   :       _FrameErr       A 1 indicates Framing Error On Reception
;
;       Bit 6   :       _unused_        Unimplemented Bit
;
;       Bit 7   :       _parityBit      The 9th bit of transmission or reception (status of PARITY bit
;                                       if parity is enabled)
;
;       To Transmit A Byte Of Data :
;                       1) Make sure _txmtProgress & _rcvOver bits are cleared
;                       2) Load TxReg with data to be transmitted
;                       3) CALL  PutChar function
;
;       To Receive A Byte Of Data :
;                       1) Make sure _txmtProgress & _rcvOver bits are cleared
;                       2) CALL GetChar function
;                       3) The received Byte is in TxReg after _rcvOver bit is cleared
;
;
; Rev 2, May 17,1994 Scott Fink
;  Corrected 7 bit and parity operation, corrected stop bit generation, corrected
;  receive prescaler settings.  Protected against inadvertant WDT reset.
;*************************************************************************************************

  Processor       16C71
  Radix   DEC
  EXPAND

  include         "16Cxx.h"

;*************************************************************************************************
;                               Setup RS-232 Parameters
;*************************************************************************************************

_ClkIn          equ     4000000         ; Input Clock Frequency is 4 Mhz
_BaudRate       set     1200            ; Baud Rate (bits per second) is 1200
_DataBits       set     8               ; 8 bit data, can be 1 to 8
_StopBits       set     1               ; 1 Stop Bit, 2 Stop Bits is not implemented

#define _PARITY_ENABLE  FALSE           ; NO Parity
#define _ODD_PARITY     FALSE           ; EVEN Parity, if Parity enabled
#define _USE_RTSCTS     FALSE           ; NO Hardware Handshaking is Used

  include         "rs232.h"

;*************************************************************************************************
;

 ORG     _ResetVector
 goto    Start
;

 ORG     _IntVector
 goto    Interrupt
;
;*************************************************************************************************
;                               Table Of ADCON0 Reg
```


© 1997 Microchip Technology Inc.

```
; Inputs : W register (valid values are 0 thru 3)
; Returns In W register, ADCON0 Value, selecting the desired Channel
;
;  Program Memory :     6 locations
;  Cycles         :     5
;
;*************************************************************************************************

GetADCon0:
 andlw   0x03                    ; mask off all bits except 2 LSBs (for Channel # 0, 1, 2, 3)
 addwf   _pcl
 retlw   (0xC1 | (0 << 3))       ; channel 0
 retlw   (0xC1 | (1 << 3))       ; channel 1
 retlw   (0xC1 | (2 << 3))       ; channel 2
GetADCon0_End:
 retlw   (0xC1 | (3 << 3))       ; channel 3

   if( (GetADCon0 & 0xff) >= (GetADCon0_End & 0xff))
 MESSG   "Warning : Crossing Page Boundary in Computed Jump, Make Sure PCLATH is Loaded Correctly"
   endif
;
;*************************************************************************************************
;                                Initialize A/D Converter
;  <RA0:RA3>    Configure as Analog Inputs, VDD as Vref
;   A/D Clock Is Internal RC Clock
;   Select Channel 0
;
;  Program Memory :     6 locations
;  Cycles         :     7
;
;*************************************************************************************************
InitAtoD:
 bsf     _rp0
 clrf    _adcon1
 bcf     _rp0
 movlw   0xC1
 movwf   _adcon0
 return
;
;*************************************************************************************************
;                                       Main Program Loop
;
;  After appropriate initilization, The main program wait for a command from the RS-232.
;  The command is 0, 1, 2 or 3. This command/data represents the A/D Channel Number.
;  After a command is received, the appropriate A/D Channel is seleted and when conversion is
;  completed the A/D Data is transmitted back to the Host. The controller now waits for a new
;  command.
;*************************************************************************************************

Start:
 call    InitSerialPort
;
WaitForNextSel:
   if _USE_RTSCTS
 bcf     _rp0
 bcf     _RTS                    ; ready to accept data from host
   endif
 call    GetChar                 ; wait for a byte reception
 btfsc   _rcvOver                ; _rcvOver Gets Cleared when a Byte Is Received (in RxReg)
 goto    $-1                     ; USER can perform other jobs here, can poll _rcvOver bit
;
; A Byte is received, Select The Desired Channel & TMXT the desired A/D Channel Data
;
 bcf     _rp0                    ; make sure to select Bank0
 movf    RxReg,w                 ; W register = Commanded Channel # (0 thru 3)
 call    GetADCon0               ; Get ADCON0 Reg Constant from Table Lookup
```

```
 movwf   _adcon0                ; Load ADCON0 reg, selecting the desired channel
 nop
;
 bsf     _go                    ; start conversion
 btfsc   _done
 goto    $-1                    ; Loop Until A/D Conversion Done

 movf    _adres,w
 movwf   TxReg
   if _USE_RTSCTS
 bsf     _RTS                   ; Half duplex mode, transmission mode, ask host not to send data
 btfsc   _CTS                   ; Check CTS signal if host ready to accept data
 goto    $-1
   endif
 call    PutChar
 btfsc   _txmtProgress
 goto    $-1                    ; Loop Until Transmission Over, User Can Perform Other Jobs


;
 goto    WaitForNextSel         ; wait for next selection (command from Serial Port)
;
;*****************************************************************************************
;                               RS-232 Routines
;*****************************************************************************************
;                               Interrupt Service Routine
;
; Only TMR0 Interrupt Is used. TMR0 Interrupt is used as timing for Serial Port Receive & Transmit
; Since RS-232 is implemented only as a Half Duplex System, The TMR0 is shared by both Receive &
; Transmit Modules.
;       Transmission :
;                       TMR0 is setup for Internal Clock increments and interrupt is
;                       generated whenTMR0 overflows. Prescaler is assigned, depending on The
;                       INPUT CLOCK & the desired BAUD RATE.
;       Reception :
;                       When put in receive mode, TMR0 is setup for external clock mode
;                       (FALLING EDGE) and preloaded with 0xFF. When a Falling Edge is
;                       detected on T0CKI Pin, TMR0 rolls over and an Interrupt is generated
;                       (thus Start Bit Detect). Once the start bit is detected, TMR0 is
;                       changed to INTERNAL CLOCK mode and TMR0 is preloaded with a certain
;                       value for regular timing interrupts to Poll T0CKI Pin (i.e RX pin).
;
;*****************************************************************************************

Interrupt:
 btfss   _rtif
 retfie                         ; other interrupt, simply return & enable GIE
;
; Save Status On INT : W register & STATUS Regs
;
 movwf   SaveWREG
 swapf   _status,w              ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
 movwf   SaveStatus
;
 btfsc   _txmtProgress
 goto    _TxmtNextBit           ; Txmt Next Bit
 btfsc   _rcvProgress
 goto    _RcvNextBit            ; Receive Next Bit
 goto    _SBitDetected          ; Must be start Bit
;
RestoreIntStatus:
 swapf   SaveStatus,w
 movwf   _status                ; restore STATUS Reg
 swapf   SaveWREG               ; save W register
 swapf   SaveWREG,w             ; restore W register
 bcf     _rtif
```


© 1997 Microchip Technology Inc.

```
 retfie
;
;*****************************************************************************************************
;
;
;
; Configure TX Pin as output, make sure TX Pin Comes up in high state on Reset
; Configure, RX_Pin (T0CKI pin) as Input, which is used to poll data on reception
;
;  Program Memory :     9 locations
;  Cycles         :     10
;*****************************************************************************************************

InitSerialPort:
 clrf    SerialStatus
;
 bcf     _rp0                    ; select Bank0 for Port Access
 bsf     TX                      ; make sure TX Pin is high on powerup, use RB Port Pullup
 bsf     _rp0                    ; Select Bank1 for TrisB access
 bcf     TX                      ; set TX Pin As Output Pin, by modifying TRIS
  if _USE_RTSCTS
 bcf     _RTS                    ; RTS is output signal, controlled by PIC16CXXX
 bsf     _CTS                    ; CTS is Input signal, controlled by the host
  endif
 bsf     RX_Pin                  ; set RX Pin As Input for reception
 return
;
;*****************************************************************************************************

 include "txmtr.asm"             ; The Transmit routines are in file "txmtr.asm"
 include "rcvr.asm"              ; The Receiver Routines are in File "rcvr.asm"

;*****************************************************************************************************

 END
```

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C

```
;*****************************************************************************************
;  GetChar Function
;  Receives a Byte Of Data
;  When reception is complete, _rcvOver Bit is cleared
;  The received data is in RxReg
;
;  Program Memory : 15 locations (17 locations if PARITY is used)
;  Cycles    : 16 (18 if PARITY is USED)
;
;*****************************************************************************************
GetChar:
 bcf      _rp0
 bsf      _rcvOver        ; Enable Reception, this bit gets reset on Byte Rcv Complete
 LOAD_BITCOUNT
 clrf     RxReg
 bcf      _FrameErr
 bcf      _ParityErr      ; Init Parity & Framing Errors
 clrf     _TMR0
 clrwdt
 bsf      _rp0
 movlw    07h
 movwf    _option
 bcf      _rp0
 clrf     _TMR0
 bsf      _rp0
 movlw    0Fh
 movwf    _option
 clrwdt
 movlw    _OPTION_SBIT    ; Inc On Ext Clk Falling Edge
 movwf    _option         ; Set Option Reg Located In Bank1
 bcf      _rp0            ; make sure to select Bank0
 movlw    0xFF
 movwf    _TMR0           ; A Start Bit will roll over TMR0 & Gen INT
 bcf      _rtif
 bsf      _rtie           ; Enable TMR0 Interrupt
 retfie                   ; Enable Global Interrupt
;
;*****************************************************************************************
;   Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
;  Program Memory :       14 locations
;  Cycles         :      12 (worst case)
;
;*****************************************************************************************
_SBitDetected:
 bcf      _rp0
 btfsc    RX_Pin          ; Make sure Start Bit Interrupt is not a Glitch
 goto     _FalseStartBit  ; False Start Bit
 bsf      _rcvProgress
 clrf     _TMR0
 clrwdt
 bsf      _rp0
 movlw    07h
 movwf    _option
 bcf      _rp0
 clrf     _TMR0
 bsf      _rp0
 movlw    0Fh
 movwf    _option
```


© 1997 Microchip Technology Inc.

```
 clrwdt
 movlw    (_BIT1_INIT | SBitPrescale)        ; Switch Back to INT Clock
 movwf    _option          ; Set Option Reg Located In Bank1
 bcf      _rp0             ; make sure to select Bank0
 LOAD_TMR0  1,(SBitTMR0Load), SBitPrescale
 goto     RestoreIntStatus
;
_FalseStartBit:
 movlw     0xFF
 movwf     _TMR0           ; reload TMR0 with 0xFF for start bit detection
 goto  RestoreIntStatus
;
;*****************************************************************************************
;    Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
;  Program Memory :       28 locations ( 43 locations with PARITY enabled)
;  Cycles         :      24 Worst Case
;
;*****************************************************************************************
_RcvNextBit:
 clrwdt
 bsf       _rp0
 movlw     07h
 movwf     _option
 bcf       _rp0
 clrf      _TMR0
 clrwdt
 bsf       _rp0
 movlw     07h
 movwf     _option
 bcf       _rp0
 clrf      _TMR0
 bsf       _rp0
 movlw     0Fh
 movwf     _option
 clrwdt
 movlw     (_OPTION_INIT | TMR0Prescale)          ; Switch Back to INT Clock
 movwf     _option           ; Set Option Reg Located In Bank1
;
 bcf       _rp0
 movf      _porta,w          ; read RX pin immediately into W register
 movwf     RxTemp
 LOAD_TMR0  0,TMR0PreLoad, TMR0Prescale ; Macro to reload TMR0
 movf      _porta,w
 xorwf     RxTemp,w
 andlw     RX_MASK           ; mask for only RX PIN (RA4)
 btfsc     _z
 goto      _PinSampled       ; both samples are same state
_SampleAgain:
 movf      _porta,w
 movwf     RxTemp            ; 2 out of 3 majority sampling done
_PinSampled:
 movf      BitCount,1
 btfsc     _z
 goto      _RcvP_Or_S
;
 decfsz    BitCount
 goto      _NextRcvBit
;
_RcvP_Or_S:
   if _PARITY_ENABLE
 decfsz    ExtraBitCount
 goto      _RcvParity
  endif
;
```


© 1997 Microchip Technology Inc.

```
_RcvStopBit:
 btfss    RX
 bsf      _FrameErr          ; may be framing Error or Glitch
 bcf      _rtie              ; disable further interrupts
 bcf      _rcvProgress
 bcf      _rcvOver           ; Byte Received, Can RCV/TXMT an other Byte
  if _PARITY_ENABLE
 movf     RxReg,w
 call     GenParity          ; Generate Parity, for Parity check
 movlw    0
 btfsc    _parityBit
 movlw    0x10               ; to mask off Received Parity Bit in _ParityErr
 xorwf    SerialStatus       ; _ParityErr bit is set accordingly
  endif
  if _DataBits == 7
 rrf      RxReg,1
 bcf      RxReg,7
  endif
 goto     RestoreIntStatus
;
_NextRcvBit:
 bcf      _carry
 btfsc    RX                 ; prepare bit for shift
 bsf      _carry
 rrf      RxReg              ; shift in received data
 goto     RestoreIntStatus
;
  if _PARITY_ENABLE
_RcvParity:
 bcf      _ParityErr         ; Temporarily store PARITY Bit in _ParityErr
 btfsc    RX                 ; Sample again to avoid any glitches
 bsf      _ParityErr
 goto     RestoreIntStatus
  endif
;*****************************************************************************************
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX D

```
;*****************************************************************************************************
;    PutChar Function
;
; Function to transmit A Byte Of Data
; Before calling this routine, load the Byte to be transmitted into TxReg
; Make sure _txmtProgress & _rcvOver bits (in Status Reg) are cleared before
; calling this routine
;
;  Program Memory :      6 locations (10 locations if PARITY is Used)
;  Cycles   :     8 (13 if PARITY is Used)
;
;*****************************************************************************************************
PutChar:
  bsf      _txmtEnable          ; enable transmission
  bsf      _txmtProgress
  LOAD_BITCOUNT                 ; Macro to load bit count
  decf     BitCount,1
  if       _DataBits == 7
  bsf      TxReg,7
  endif
;
  if _PARITY_ENABLE
  movf     TxReg,W
  call     GenParity           ; If Parity is used, then Generate Parity Bit
  endif
;
  call     _TxmtStartBit
  bsf      _rtie               ; Enable TMR0 Overflow INT
  retfie                       ; return with _GIE Bit Set
;
;*****************************************************************************************************
;   Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
;  Program Memory      :   30 locations  (38 locations if PARITY is used)
;  Cycles              :   15 Worst Case
;
;*****************************************************************************************************

_TxmtNextBit:
  bcf      _rp0
  LOAD_TMR0  0,TMR0PreLoad, TMR0Prescale            ; Macro to reload TMR0
;
  movf     BitCount            ; done with data xmission?
  btfsc    _z
  goto     _ParityOrStop       ; yes, do parity or stop bit
;
  decf     BitCount
  goto     _NextTxmtBit        ; no, send another
;
_ParityOrStop:
  if _PARITY_ENABLE
  btfsc    ExtraBitCount,1     ; ready for parity bit?
  goto     _SendParity
  endif
  movf     ExtraBitCount,1     ; check if sending stop bit
  btfsc    _z
  goto     DoneTxmt
  decf     ExtraBitCount,1
;
```

```
_StopBit:
  bsf      TX                  ; STOP Bit is High
  goto     RestoreIntStatus
  goto     DoneTxmt
;
_NextTxmtBit:
  bsf      _carry
  rrf      TxReg
  btfss    _carry
  bcf      TX
  btfsc    _carry
  bsf      TX
;
  btfss    _txmtEnable
  bsf      _rtie               ; disable further interrupts, Transmission Aborted
;
  goto     RestoreIntStatus
;
  if       _PARITY_ENABLE
_SendParity:
  decf     ExtraBitCount,1     ; subtract parity from count
  btfss    _parityBit
  bcf      TX
  btfsc    _parityBit
  bsf      TX
  goto     RestoreIntStatus
  endif

DoneTxmt
  bsf      TX                  ; STOP Bit is High
  bcf      _rtie               ; disable further interrupts
  bcf      _txmtProgress       ; indicates end of xmission
  goto     RestoreIntStatus
;
;*****************************************************************************************************
;   Internal Subroutine
; entered from Interrupt Service Routine when Start Bit Is detected.
;
;  Program Memory     : 9 locations
;  Cycles             : 10
;
;*****************************************************************************************************
_TxmtStartBit:
  bcf      _rp0
  clrf     _TMR0
  clrwdt
  bsf      _rp0
  movlw    07h
  movwf    _option
  bcf      _rp0
  clrf     _TMR0
  bsf      _rp0
  movlw    0Fh
  movwf    _option
  clrwdt
  movlw    (_OPTION_INIT | TMR0Prescale)
  movwf    _option             ; Set Option Reg Located In Bank1
  bcf      _rp0                ; make sure to select Bank0
  bcf      TX                  ; Send Start Bit
  movlw    -TMR0PreLoad        ; Prepare for Timing Interrupt
  movwf    _TMR0
  bcf      _rtif
  return

;*****************************************************************************************************
;    Generate Parity for the Value in W register
```


© 1997 Microchip Technology Inc.

```
;
;  The parity bit is set in _parityBit (SerialStatus,7)
;  Common Routine For Both Transmission & Reception
;
;  Program Memory      : 16 locations
;  Cycles              : 72
;
;*****************************************************************************************************
  if _PARITY_ENABLE

GenParity:
  movwf    temp2              ; save data
  movf     BitCount,w         ; save bitcount
  movwf    temp1
Parityloop
  rrf      temp2
  btfss    _carry             ; put data in carry bit
  goto     NotOne
  xorlw    00h                ; parity calculated by XORing all data bits
  goto     OneDone
NotOne
  xorlw    01h
OneDone
  decfsz   temp1
  goto     Parityloop         ; decrement count
  movwf    temp1
; Parity bit is in Bit 0 of temp1
;
    if _ODD_PARITY
  bsf      _parityBit
  btfsc    temp1,0
  bcf      _parityBit
    else
  bcf      _parityBit
  btfsc    temp1,0
  bsf      _parityBit
    endif

  return
  endif
;*****************************************************************************************************
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX E

```
MPASM 01.00.02 Alpha  \PICMASTR\CU   5-20-1994  9:13:56                PAGE  1
RS232 Communications : Half Duplex : PIC16C6x/7x/8x
Software Implementation : Interrupt Driven
LOC  OBJECT CODE     LINE SOURCE TEXT

                     0001         TITLE           "RS232 Communications : Half Duplex : PIC16C6x/7x/8x"
                     0002         SUBTITLE        "Software Implementation : Interrupt Driven"
                     0003
                     0004 ;*****************************************************************************************************
                     0005 ;                        Software Implementation Of RS232 Communications Using PIC16CXXX
                     0006 ;                                          Half-Duplex
                     0007 ;
                     0008 ;  These routines are intended to be used with PIC16C6X/7X family. These routines can be
                     0009 ;  used with processors in the 16C6X/7X family which do not have on board Hardware Async
                     0010 ;  Serial Port.
                     0011 ;  MX..
                     0012 ;
                     0013 ;  Description :
                     0014 ;                Half Duplex RS-232 Mode Is implemented in Software.
                     0015 ;                Both Reception & Transmission are Interrupt driven
                     0016 ;                Only 1 peripheral (Timer0) used for both transmission & reception
                     0017 ;                TimerR0 is used for both timing generation (for bit transmission & bit polling)
                     0018 ;                and Start Bit Detection in reception mode.
                     0019 ;                This is explained in more detail in the Interrupt Subroutine.
                     0020 ;                Programmable Baud Rate (speed depnding on Input Clock Freq.), programmable
                     0021 ;                #of bits, Parity enable/disable, odd/even parity is implemented.
                     0022 ;                        Parity & Framing errors are detected on Reception
                     0023 ;
                     0024 ;                                  RS-232 Parameters
                     0025 ;
                     0026 ;The RS-232 Parameters are defined as shown below:
                     0027 ;
                     0028 ;                _ClkIn          :       Input Clock Frequency of the processor
                     0029 ;                                        (NOTE : RC Clock Mode Is Not Suggested due to wide variations)
                     0030 ;                _BaudRate       :       Desired Baud Rate. Any valid value can be used.
                     0031 ;                                        The highest Baud Rate achievable depends on Input Clock Freq.
                     0032 ;                                        300 to 4800 Baud was tested using 4 Mhz Input Clock
                     0033 ;                                        300 to 19200 Baud was tested using 10 Mhz Input Clock
                     0034 ;                                        Higher rates can be obtained using higher Input Clock Frequencies.
                     0035 ;                                        Once the _BaudRate & _ClkIn are specified the program
                     0036 ;                                        automatically selectes all the appropiate timings
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
0037 ;              _DataBits       :     Can specify 1 to 8 Bits.
0038 ;              _StopBits       :     Limited to 1 Stop Bit. Must set it to 1.
0039 ;              _PARITY_ENABLE  :     Parity Enable Flag. Set it to TRUE or FALSE. If PARITY
0040 ;                                    is used, then set it to TRUE, else FALSE. See "_ODD_PARITY" flag
0041 ;                                    description below
0042 ;              _ODD_PARITY     :     Set it to TRUE or FALSE. If TRUE, then ODD PARITY is used, else
0043 ;                                    EVEN Parity Scheme is used.
0044 ;                                    This Flag is ignored if _PARITY_ENABLE is set to FALSE.
0045 ;
0046 ;
0047 ;  Usage :
0048 ;              An example is given in the main program on how to Receive & Transmit Data. In the
0049 ;              example, the processor waits until a command is received. The command is interpreted as
0050 ;              the A/D Channel Number of PIC16C71. Upon reception of a command, the desired A/D channel
0051 ;              is selected and after A/D conversion, the 8 Bit A/D data is transmitted back to the Host.
0052 ;
0053 ;                       The RS-232 Control/Status Reg's bits are explained below :
0054 ;
0055 ;       "SerialStatus"          : RS-232 Status/Control Register
0056 ;
0057 ;       Bit 0   :       _txmtProgress   (1 if transmission in progress, 0 if transmission is complete)
0058 ;                                       After a byte is transmitted by calling "PutChar" function, the
0059 ;                                       user's code can poll this bit to check if transmission is
0060 ;                                       complete. This bit is reset after STOP bit has been transmitted.
0061 ;       Bit 1   :       _txmtEnable     Set this bit to 1 on initialization to enable transmission.
0062 ;                                       This bit can be used to Abort a transmission while the
0063 ;                                       transmitter is in progress (i.e when _txmtProgress = 1)
0064 ;       Bit 2   :       _rcvProgress    Indicates that the receiver is in middle of reception. It is
0065 ;                                       reset when a byte is received.
0066 ;       Bit 3   :       _rcvOver        This bit indicates the completion of Reception of a Byte. The
0067 ;                                       user's ode can poll this bit after calling "GetChar" function.
0068 ;                                       Once "GetChar"is called, this bit is 1 and cleared to 0 after
0069 ;                                       reception of a complete byte (parity bit if enabled & stop bit)
0070 ;       Bit 4   :       _ParityErr      A 1 indicates Parity Error on Reception (both even & odd parity)
0071 ;       Bit 5   :       _FrameErr       A 1 indicates Framing Error On Reception
0072 ;
0073 ;       Bit 6   :       _unused_        Unimplemented Bit
0074 ;
0075 ;       Bit 7   :       _parityBit      The 9 th bit of transmission or reception (status of PARITY bit
0076 ;                                       if parity is enabled)
0077 ;
0078 ;       To Transmit A Byte Of Data :
0079 ;                       1) Make sure _txmtProgress & _rcvOver bits are cleared
0080 ;                       2) Load TxReg with data to be transmitted
0081 ;                       3) CALL  PutChar function
0082 ;
0083 ;       To Receive A Byte Of Data :
```

```
0084 ;                      1) Make sure _txmtProgress & _rcvOver bits are cleared
0085 ;                      2) CALL GetChar function
0086 ;                      3) The received Byte is in TxReg after _rcvOver bit is cleared
0087 ;
0088 ;
0089 ;   Rev 2, May 17,1994 Scott Fink
0090 ;          Corrected 7 bit and parity operation, corrected stop bit generation, corrected
0091 ;          receive prescaler settings.  Protected against inadvertant WDT reset.
0092 ;*****************************************************************************************************
0093
0094                Processor       16C71
0095                Radix   DEC
0096                EXPAND
0097
0098            include         "16Cxx.h"
0179
0180
0181
0098
0099
0100 ;*****************************************************************************************************
0101 ;                               Setup RS-232 Parameters
0102 ;*****************************************************************************************************
0103
003D 0900       0104 _ClkIn          equ      4000000         ; Input Clock Frequency is 4 Mhz
04B0            0105 _BaudRate       set      1200            ; Baud Rate (bits per second) is 1200
0008            0106 _DataBits       set      8               ; 8 bit data, can be 1 to 8
0001            0107 _StopBits       set      1               ; 1 Stop Bit, 2 Stop Bits is not implemented
                0108
0046            0109  #define _PARITY_ENABLE  FALSE          ; NO Parity
0047            0110  #define _ODD_PARITY     FALSE          ; EVEN Parity, if Parity enabled
0048            0111  #define _USE_RTSCTS     FALSE          ; NO Hardware Handshaking is Used
                0112
0113            0113            include         "rs232.h"
0001            0114
0113            0115 ;*****************************************************************************************************
0114            0116 ;
0115            0117
0116            0118         ORG     _ResetVector
0000 2811       0119         goto    Start
0117            0120 ;
0118            0121
0119            0122         ORG     _IntVector
0004 2823       0123         goto    Interrupt
0120            0124 ;
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
                        0125 ;*****************************************************************************************************
                        0126 ;                                   Table Of ADCON0 Reg
                        0127 ; Inputs : W register (valid values are 0 thru 3)
                        0128 ; Returns In W register, ADCON0 Value, selecting the desired Channel
                        0129 ;
                        0130 ;  Program Memory :      6 locations
                        0131 ;  Cycles         :      5
                        0132 ;
                        0133 ;*****************************************************************************************************
                        0134
                        0135 GetADCon0:
0005 3903               0136         andlw   0x03                    ; mask off all bits except 2 LSBs (for Channel # 0, 1, 2, 3)
0006 0782               0137         addwf   _pcl
0007 34C1               0138         retlw   (0xC1 | (0 << 3))       ; channel 0
0008 34C9               0139         retlw   (0xC1 | (1 << 3))       ; channel 1
0009 34D1               0140         retlw   (0xC1 | (2 << 3))       ; channel 2
                        0141 GetADCon0_End:
000A 34D9               0142         retlw   (0xC1 | (3 << 3))       ; channel 3
                        0143
                        0144    if( (GetADCon0 & 0xff) >= (GetADCon0_End & 0xff))
                        0145         MESSG   "Warning : Crossing Page Boundary in Computed Jump, Make Sure PCLATH is Loaded Correctly"
                        0146    endif
                        0147 ;
                        0148 ;*****************************************************************************************************
                        0149 ;                                   Initialize A/D Converter
                        0150 ;  <RA0:RA3>    Configure as Analog Inputs, VDD as Vref
                        0151 ;   A/D Clock Is Internal RC Clock
                        0152 ;   Select Channel 0
                        0153 ;
                        0154 ;  Program Memory :      6 locations
                        0155 ;  Cycles         :      7
                        0156 ;
                        0157 ;*****************************************************************************************************
                        0158 InitAtoD:
000B 1683               0159         bsf     _rp0
000C 0188               0160         clrf    _adcon1
000D 1283               0161         bcf     _rp0
000E 30C1               0162         movlw   0xC1
000F 0088               0163         movwf   _adcon0
0010 0008               0164         return
                        0165 ;
                        0166 ;*****************************************************************************************************
                        0167 ;                                       Main Program Loop
                        0168 ;
                        0169 ;  After appropriate initialization, The main program wait for a command from RS-232
                        0170 ;  The command is 0, 1, 2 or 3. This command/data represents the A/D Channel Number.
                        0171 ;  After a command is received, the appropriate A/D Channel is selected and when conversion is
```

```
                0172 ;  completed the A/D Data is transmitted back to the Host. The controller now waits for a new
                0173 ;  command.
                0174 ;*****************************************************************************************************
                0175
                0176 Start:
0011 2033       0177        call    InitSerialPort
                0178 ;
                0179 WaitForNextSel:
                0180    if _USE_RTSCTS
                0181        bcf     _rp0
                0182        bcf     _RTS                ; ready to accept data from host
                0183    endif
0012 2074       0184        call    GetChar             ; wait for a byte reception
0013 198F       0185        btfsc   _rcvOver            ; _rcvOver Gets Cleared when a Byte Is Received (in RxReg)
0014 2813       0186        goto    $-1                 ; USER can perform other jobs here, can poll _rcvOver bit
                0187 ;
                0188 ; A Byte is received, Select The Desired Channel & TMXT the desired A/D Channel Data
                0189 ;
0015 1283       0190        bcf     _rp0                ; make sure to select Bank0
0016 080D       0191        movf    RxReg,w             ; W register = Commanded Channel # (0 thru 3)
0017 2005       0192        call    GetADCon0           ; Get ADCON0 Reg Constant from Table Lookup
0018 0088       0193        movwf   _adcon0             ; Load ADCON0 reg, selecting the desired channel
0019 0000       0194        nop
                0195 ;
001A 1508       0196        bsf     _go                 ; start conversion
001B 1908       0197        btfsc   _done
001C 281B       0198        goto    $-1                 ; Loop Until A/D Conversion Done
                0199
001D 0809       0200        movf    _adres,w
001E 008C       0201        movwf   TxReg
                0202    if _USE_RTSCTS
                0203        bsf     _RTS                ; Half duplex mode, transmission mode, ask host not to send data
                0204        btfsc   _CTS                ; Check CTS signal if host ready to accept data
                0205        goto    $-1
                0206    endif
ww001F 203A     0207        call    PutChar
0020 180F       0208        btfsc   _txmtProgress
0021 2820       0209        goto    $-1                 ; Loop Until Transmission Over, User Can Perform Other Jobs
                0210
                0211
                0212 ;
0022 2812       0213        goto    WaitForNextSel      ; wait for next selection (command from Serial Port)
                0214 ;
                0215 ;*****************************************************************************************************
                0216 ;                               RS-232 Routines
                0217 ;*****************************************************************************************************
                0218 ;                               Interrupt Service Routine
```


© 1997 Microchip Technology Inc.

```
                        0219 ;
                        0220 ; Only TMR0 Inturrupt Is used. TMR0 Interrupt is used as timing for Serial Port Receive & Transmit
                        0221 ; Since RS-232 is implemented only as a Half Duplex System, Timer0 is shared by both Receive &
                        0222 ; Transmit Modules.
                        0223 ;       Transmission :
                        0224 ;                       Timer0 is setup for Internal Clock increments and interrupt is generated when
                        0225 ;                       TMR0 overflows. Prescaler is assigned, depending on The INPUT CLOCK & the
                        0226 ;                       desired BAUD RATE.
                        0227 ;          Reception :
                        0228 ;                       When put in receive mode, Timer0 is setup for external clock mode (FALLING EDGE)
                        0229 ;                       and preloaded with 0xFF. When a Falling Edge is detected on T0CKI Pin, TMR0
                        0230 ;                       rolls over and an Interrupt is generated (thus Start Bit Detect). Once the start
                        0231 ;                       bit is detected, TIMER0 is changed to INTERNAL CLOCK mode and TMR0 is preloaded
                        0232 ;                       with a certain value for regular timing interrupts to Poll T0CKI Pin (i.e RX pin).
                        0233 ;
                        0234 ;*****************************************************************************************************
                        0235
                        0236 Interrupt:
0023 1D0B               0237         btfss   _rtif
0024 0009               0238         retfie                          ; other interrupt, simply return & enable GIE
                        0239 ;
                        0240 ; Save Status On INT : W register & STATUS Regs
                        0241 ;
0025 0092               0242         movwf   SaveWREG
0026 0E03               0243         swapf   _status,w               ; affects no STATUS bits : Only way OUT to save STATUS Reg ?????
0027 0093               0244         movwf   SaveStatus
                        0245 ;
0028 180F               0246         btfsc   _txmtProgress
0029 2844               0247         goto    _TxmtNextBit            ; Txmt Next Bit
002A 190F               0248         btfsc   _rcvProgress
002B 28A8               0249         goto    _RcvNextBit             ; Receive Next Bit
002C 2890               0250         goto    _SBitDetected           ; Must be start Bit
                        0251 ;
                        0252 RestoreIntStatus:
002D 0E13               0253         swapf   SaveStatus,w
002E 0083               0254         movwf   _status                 ; restore STATUS Reg
002F 0E92               0255         swapf   SaveWREG                ; save W register
0030 0E12               0256         swapf   SaveWREG,w              ; restore W register
0031 110B               0257         bcf     _rtif
0032 0009               0258         retfie
                        0259 ;
                        0260 ;*****************************************************************************************************
                        0261 ;
                        0262 ;
                        0263 ;
                        0264 ; Configure TX Pin as output, make sure TX Pin Comes up in high state on Reset
                        0265 ; Configure, RX_Pin (T0CKI pin) as Input, which is used to poll data on reception
```

© 1997 Microchip Technology Inc.

```
                    0266 ;
                    0267 ;  Program Memory :     9 locations
                    0268 ;  Cycles         :     10
                    0269 ;*****************************************************************************************************
                    0270
                    0271 InitSerialPort:
0033 018F           0272         clrf    SerialStatus
                    0273 ;
0034 1283           0274         bcf     _rp0                        ; select Bank0 for Port Access
0035 1786           0275         bsf     TX                          ; make sure TX Pin is high on powerup, use RB Port Pullup
0036 1683           0276         bsf     _rp0                        ; Select Bank1 for TrisB access
0037 1386           0277         bcf     TX                          ; set TX Pin As Output Pin, by modifying TRIS
                    0278   if _USE_RTSCTS
                    0279         bcf     _RTS                        ; RTS is output signal, controlled by PIC16CXXX
                    0280         bsf     _CTS                        ; CTS is Input signal, controlled by the host
                    0281   endif
0038 1605           0282         bsf     RX_Pin                      ; set RX Pin As Input for reception
0039 0008           0283         return
                    0284 ;
                    0285 ;*****************************************************************************************************
                    0286
                    0287       include "txmtr.asm"           ; The Transmit routines are in file "txmtr.asm"
                    0001 ;*****************************************************************************************************
                    0002 ;     PutChar Function
                    0003 ;
                    0004 ;     Function to transmit A Byte Of Data
                    0005 ;     Before calling this routine, load the Byte to be transmitted into TxReg
                    0006 ;     Make sure _txmtProgress & _rcvOver bits (in Status Reg) are cleared before
                    0007 ;     calling this routine
                    0008 ;
                    0009 ;  Program Memory :  6 locations (10 locations if PARITY is Used)
                    0010 ;  Cycles  :   8 (13 if PARITY is Used)
                    0011 ;
                    0012 ;*****************************************************************************************************
                    0013 PutChar:
003A 148F           0014         bsf     _txmtEnable                 ; enable transmission
003B 140F           0015         bsf     _txmtProgress
                    0016     LOAD_BITCOUNT                           ; Macro to load bit count
003C 3009              M         movlw   _DataBits+1
003D 0090              M         movwf   BitCount
003E 3001              M         movlw   1
003F 0091              M         movwf   ExtraBitCount
                       M     if  _PARITY_ENABLE
                       M         movlw   2
                       M         movwf   ExtraBitCount
                       M     endif
0040 0390           0017         decf    BitCount,1
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
                0018   if _DataBits == 7
                0019         bsf     TxReg,7
                0020   endif
                0021 ;
                0022    if _PARITY_ENABLE
                0023         movf    TxReg,W
                0024         call    GenParity                  ; If Parity is used, then Generate Parity Bit
                0025    endif
                0026 ;
0041 2060       0027         call    _TxmtStartBit
0042 168B       0028         bsf     _rtie                      ; Enable TMR0 Overflow INT
0043 0009       0029         retfie                             ; return with _GIE Bit Set
                0030 ;
                0031 ;*************************************************************************************************
                0032 ;                    Internal Subroutine
                0033 ; entered from Interrupt Service Routine when Start Bit Is detected.
                0034 ;
                0035 ;  Program Memory :   30 locations  (38 locations if PARITY is used)
                0036 ;  Cycles    :   15 Worst Case
                0037 ;
                0038 ;*************************************************************************************************
                0039
                0040 _TxmtNextBit:
0044 1283       0041         bcf     _rp0
                0042     LOAD_TMR0  0,TMR0PreLoad, TMR0Prescale      ; Macro to reload TMR0
                   M     if(UsePrescale == 0 && 0 == 0)
                   M         movlw   -TMR0PreLoad + _Cycle_Offset1
                   M     else
0045 3036          M         movlw   -TMR0PreLoad + (_Cycle_Offset1 >> (TMR0Prescale+1))  ; Re Load TMR0 init value + INT La
                   M     endif
0046 0081          M         movwf   _TMR0                      ; Note that Prescaler is cleared when TMR0 is written
                0043 ;
0047 0890       0044         movf    BitCount                   ;done with data xmission?
0048 1903       0045         btfsc   _z
0049 284C       0046         goto    _ParityOrStop              ;yes, do parity or stop bit
                0047 ;
004A 0390       0048         decf    BitCount
004B 2853       0049         goto    _NextTxmtBit               ;no, send another
                0050 ;
                0051 _ParityOrStop:
                0052     if _PARITY_ENABLE
                0053         btfsc   ExtraBitCount,1            ;ready for parity bit?
                0054         goto    _SendParity
                0055     endif
004C 0891       0056         movf    ExtraBitCount,1            ;check if sending stop bit
004D 1903       0057         btfsc   _z
004E 285C       0058         goto    DoneTxmt
```

```
004F 0391          0059         decf    ExtraBitCount,1
                   0060 ;
                   0061 _StopBit:
0050 1786          0062         bsf     TX                          ; STOP Bit is High
0051 282D          0063         goto    RestoreIntStatus
0052 285C          0064         goto    DoneTxmt
                   0065 ;
                   0066 _NextTxmtBit:
0053 1403          0067         bsf     _carry
0054 0C8C          0068         rrf     TxReg
0055 1C03          0069         btfss   _carry
0056 1386          0070         bcf     TX
0057 1803          0071         btfsc   _carry
0058 1786          0072         bsf     TX
                   0073 ;
0059 1C8F          0074         btfss   _txmtEnable
005A 168B          0075         bsf     _rtie                       ; disable further interrupts, Transmission Aborted
                   0076 ;
005B 282D          0077         goto    RestoreIntStatus
                   0078 ;
                   0079   if _PARITY_ENABLE
                   0080 _SendParity:
                   0081         decf    ExtraBitCount,1             ;subtract parity from count
                   0082         btfss   _parityBit
                   0083         bcf     TX
                   0084         btfsc   _parityBit
                   0085         bsf     TX
                   0086         goto    RestoreIntStatus
                   0087   endif
                   0088
                   0089 DoneTxmt
005C 1786          0090         bsf     TX                          ;STOP Bit is High
005D 128B          0091         bcf     _rtie                       ;disable further interrupts
005E 100F          0092         bcf     _txmtProgress               ;indicates end of xmission
005F 282D          0093         goto    RestoreIntStatus
                   0094 ;
                   0095 ;*************************************************************************************************
                   0096 ;                  Internal Subroutine
                   0097 ; entered from Interrupt Service Routine when Start Bit Is detected.
                   0098 ;
                   0099 ;  Program Memory :     9 locations
                   0100 ;  Cycles  :       10
                   0101 ;
                   0102 ;*************************************************************************************************
                   0103 _TxmtStartBit:
0060 1283          0104         bcf     _rp0
0061 0181          0105         clrf    _TMR0
```


© 1997 Microchip Technology Inc.

```
0062 0064           0106         clrwdt
0063 1683           0107         bsf     _rp0
0064 3007           0108         movlw   07h
0065 0081           0109         movwf   _option
0066 1283           0110         bcf     _rp0
0067 0181           0111         clrf    _TMR0
0068 1683           0112         bsf     _rp0
0069 300F           0113         movlw   0Fh
006A 0081           0114         movwf   _option
006B 0064           0115         clrwdt
006C 3001           0116         movlw   (_OPTION_INIT | TMR0Prescale)
006D 0081           0117         movwf   _option                         ; Set Option Reg Located In Bank1
006E 1283           0118         bcf     _rp0                            ; make sure to select Bank0
006F 1386           0119         bcf     TX                              ; Send Start Bit
0070 3030           0120         movlw   -TMR0PreLoad                    ; Prepare for Timing Interrupt
0071 0081           0121         movwf   _TMR0
0072 110B           0122         bcf     _rtif
0073 0008           0123         return
                    0124
                    0125 ;*****************************************************************************************************
                    0126 ;                    Generate Parity for the Value in W register
                    0127 ;
                    0128 ;  The parity bit is set in _parityBit (SerialStatus,7)
                    0129 ;  Common Routine For Both Transmission & Reception
                    0130 ;
                    0131 ;  Program Memory :     16 locations
                    0132 ;  Cycles  :       72
                    0133 ;
                    0134 ;*****************************************************************************************************
                    0135   if _PARITY_ENABLE
                    0136
                    0137 GenParity:
                    0138         movwf   temp2                          ;save data
                    0139         movf    BitCount,w                     ;save bitcount
                    0140         movwf   temp1
                    0141 Parityloop
                    0142         rrf     temp2
                    0143         btfss   _carry                         ;put data in carry bit
                    0144         goto    NotOne
                    0145         xorlw   00h                            ;parity calculated by XORing all data bits
                    0146         goto    OneDone
                    0147 NotOne
                    0148         xorlw   01h
                    0149 OneDone
                    0150         decfsz  temp1
                    0151         goto    Parityloop                     ;decrement count
                    0152         movwf   temp1
```

© 1997 Microchip Technology Inc.

```
                    0153 ; Parity bit is in Bit 0 of temp1
                    0154 ;
                    0155     if _ODD_PARITY
                    0156         bsf     _parityBit
                    0157         btfsc   temp1,0
                    0158         bcf     _parityBit
                    0159     else
                    0160         bcf     _parityBit
                    0161         btfsc   temp1,0
                    0162         bsf     _parityBit
                    0163     endif
                    0164
                    0165         return
                    0166   endif
                    0167 ;*****************************************************************************************************
                    0287
                    0288     include "rcvr.asm"       ; The Receiver Routines are in File "rcvr.asm"
                    0001 ;*****************************************************************************************
                    0002 ;                   GetChar Function
                    0003 ;  Receives a Byte Of Data
                    0004 ;          When reception is complete, _rcvOver Bit is cleared
                    0005 ;          The received data is in RxReg
                    0006 ;
                    0007 ;  Program Memory :     15 locations (17 locations if PARITY is used)
                    0008 ;  Cycles    :     16 (18 if PARITY is USED)
                    0009 ;
                    0010 ;*****************************************************************************************
                    0011 GetChar:
0074 1283           0012         bcf     _rp0
0075 158F           0013         bsf     _rcvOver                    ; Enable Reception, this bit gets reset on Byte Rcv Complete
                    0014     LOAD_BITCOUNT
0076 3009              M         movlw   _DataBits+1
0077 0090              M         movwf   BitCount
0078 3001              M         movlw   1
0079 0091              M         movwf   ExtraBitCount
                       M     if  _PARITY_ENABLE
                       M         movlw   2
                       M         movwf   ExtraBitCount
                       M     endif
007A 018D           0015         clrf    RxReg
007B 128F           0016         bcf     _FrameErr
007C 120F           0017         bcf     _ParityErr                  ; Init Parity & Framing Errors
007D 0181           0018         clrf    _TMR0
007E 0064           0019         clrwdt
007F 1683           0020         bsf     _rp0
0080 3007           0021         movlw   07h
0081 0081           0022         movwf   _option
```


© 1997 Microchip Technology Inc.

```
0082 1283           0023         bcf     _rp0
0083 0181           0024         clrf    _TMR0
0084 1683           0025         bsf     _rp0
0085 300F           0026         movlw   0Fh
0086 0081           0027         movwf   _option
0087 0064           0028         clrwdt
0088 3038           0029         movlw   _OPTION_SBIT          ; Inc On Ext Clk Falling Edge
0089 0081           0030         movwf   _option               ; Set Option Reg Located In Bank1
008A 1283           0031         bcf     _rp0                  ; make sure to select Bank0
008B 30FF           0032         movlw   0xFF
008C 0081           0033         movwf   _TMR0                 ; A Start Bit will roll over TMR0 & Gen INT
008D 110B           0034         bcf     _rtif
008E 168B           0035         bsf     _rtie                 ; Enable TMR0 Interrupt
008F 0009           0036         retfie                        ; Enable Global Interrupt
                    0037 ;
                    0038 ;*************************************************************************************
                    0039 ;                   Internal Subroutine
                    0040 ; entered from Interrupt Service Routine when Start Bit Is detected.
                    0041 ;
                    0042 ;  Program Memory :     14 locations
                    0043 ;  Cycles    :     12 (worst case)
                    0044 ;
                    0045 ;*************************************************************************************
                    0046 _SBitDetected:
0090 1283           0047         bcf     _rp0
0091 1A05           0048         btfsc   RX_Pin                        ; Make sure Start Bit Interrupt is not a Glitch
0092 28A5           0049         goto    _FalseStartBit                ; False Start Bit
0093 150F           0050         bsf     _rcvProgress
0094 0181           0051         clrf    _TMR0
0095 0064           0052         clrwdt
0096 1683           0053         bsf     _rp0
0097 3007           0054         movlw   07h
0098 0081           0055         movwf   _option
0099 1283           0056         bcf     _rp0
009A 0181           0057         clrf    _TMR0
009B 1683           0058         bsf     _rp0
009C 300F           0059         movlw   0Fh
009D 0081           0060         movwf   _option
009E 0064           0061         clrwdt
009F 3002           0062         movlw   (_BIT1_INIT | SBitPrescale)   ; Switch Back to INT Clock
00A0 0081           0063         movwf   _option                       ; Set Option Reg Located In Bank1
00A1 1283           0064         bcf     _rp0                          ; make sure to select Bank0
                    0065     LOAD_TMR0  1,(SBitTMR0Load), SBitPrescale
                       M     if(UsePrescale == 0 && 1 == 0)
                       M         movlw   -(SBitTMR0Load) + _Cycle_Offset1
                       M     else
00A2 3081              M         movlw   -(SBitTMR0Load) + (_Cycle_Offset1 >> (SBitPrescale+1))  ; Re Load TMR0 init value + INT La
```

© 1997 Microchip Technology Inc.

```
                    M     endif
00A3 0081           M         movwf   _TMR0           ; Note that Prescaler is cleared when TMR0 is written
00A4 282D      0066         goto    RestoreIntStatus
               0067 ;
               0068 _FalseStartBit:
00A5 30FF      0069         movlw   0xFF
00A6 0081      0070         movwf   _TMR0                           ; reload TMR0 with 0xFF for start bit detection
00A7 282D      0071         goto    RestoreIntStatus
               0072 ;
               0073 ;*************************************************************************************
               0074 ;                   Internal Subroutine
               0075 ; entered from Interrupt Service Routine when Start Bit Is detected.
               0076 ;
               0077 ;  Program Memory :     28 locations ( 43 locations with PARITY enabled)
               0078 ;  Cycles  :       24 Worst Case
               0079 ;
               0080 ;*************************************************************************************
               0081 _RcvNextBit:
00A8 0064      0082         clrwdt
00A9 1683      0083         bsf     _rp0
00AA 3007      0084         movlw   07h
00AB 0081      0085         movwf   _option
00AC 1283      0086         bcf     _rp0
00AD 0181      0087         clrf    _TMR0
00AE 0064      0088         clrwdt
00AF 1683      0089         bsf     _rp0
00B0 3007      0090         movlw   07h
00B1 0081      0091         movwf   _option
00B2 1283      0092         bcf     _rp0
00B3 0181      0093         clrf    _TMR0
00B4 1683      0094         bsf     _rp0
00B5 300F      0095         movlw   0Fh
00B6 0081      0096         movwf   _option
00B7 0064      0097         clrwdt
00B8 3001      0098         movlw   (_OPTION_INIT | TMR0Prescale)   ; Switch Back to INT Clock
00B9 0081      0099         movwf   _option                         ; Set Option Reg Located In Bank1
               0100 ;
00BA 1283      0101         bcf     _rp0
00BB 0805      0102         movf    _porta,w                        ; read RX pin immediately into W register
00BC 008E      0103         movwf   RxTemp
               0104     LOAD_TMR0  0,TMR0PreLoad, TMR0Prescale      ; Macro to reload TMR0
                  M     if(UsePrescale == 0 && 0 == 0)
                  M         movlw   -TMR0PreLoad + _Cycle_Offset1
                  M     else
00BD 3036         M         movlw   -TMR0PreLoad + (_Cycle_Offset1 >> (TMR0Prescale+1))  ; Re Load TMR0 init value + INT La
                  M     endif
00BE 0081         M         movwf   _TMR0                           ; Note that Prescaler is cleared when TMR0 is written
```


© 1997 Microchip Technology Inc.

```
00BF 0805          0105          movf    _porta,w
00C0 060E          0106          xorwf   RxTemp,w
00C1 3910          0107          andlw   RX_MASK                 ; mask for only RX PIN (RA4)
00C2 1903          0108          btfsc   _z
00C3 28C6          0109          goto    _PinSampled             ; both samples are same state
                   0110 _SampleAgain:
00C4 0805          0111          movf    _porta,w
00C5 008E          0112          movwf   RxTemp                  ; 2 out of 3 majority sampling done
                   0113 _PinSampled:
00C6 0890          0114          movf    BitCount,1
00C7 1903          0115          btfsc   _z
00C8 28CB          0116          goto    _RcvP_Or_S
                   0117 ;
00C9 0B90          0118          decfsz  BitCount
00CA 28D1          0119          goto    _NextRcvBit
                   0120 ;
                   0121 _RcvP_Or_S:
                   0122    if _PARITY_ENABLE
                   0123          decfsz  ExtraBitCount
                   0124          goto    _RcvParity
                   0125    endif
                   0126 ;
                   0127 _RcvStopBit:
00CB 1E0E          0128          btfss   RX
00CC 168F          0129          bsf     _FrameErr               ; may be framing Error or Glitch
00CD 128B          0130          bcf     _rtie                   ; disable further interrupts
00CE 110F          0131          bcf     _rcvProgress
00CF 118F          0132          bcf     _rcvOver                ; Byte Received, Can RCV/TXMT an other Byte
                   0133   if _PARITY_ENABLE
                   0134          movf    RxReg,w
                   0135          call    GenParity               ; Generate Parity, for Parity check
                   0136          movlw   0
                   0137          btfsc   _parityBit
                   0138          movlw   0x10                    ; to mask off Received Parity Bit in _ParityErr
                   0139          xorwf   SerialStatus            ; _ParityErr bit is set accordingly
                   0140   endif
                   0141   if _DataBits == 7
                   0142          rrf     RxReg,1
                   0143          bcf     RxReg,7
                   0144   endif
00D0 282D          0145          goto    RestoreIntStatus
                   0146 ;
                   0147 _NextRcvBit:
00D1 1003          0148          bcf     _carry
00D2 1A0E          0149          btfsc   RX                      ; prepare bit for shift
00D3 1403          0150          bsf     _carry
00D4 0C8D          0151          rrf     RxReg                   ; shift in received data
```

© 1997 Microchip Technology Inc.

```
00D5 282D           0152         goto    RestoreIntStatus
                    0153 ;
                    0154   if _PARITY_ENABLE
                    0155 _RcvParity:
                    0156         bcf     _ParityErr                  ; Temporarily store PARITY Bit in _ParityErr
                    0157         btfsc   RX                          ; Sample again to avoid any glitches
                    0158         bsf     _ParityErr
                    0159         goto    RestoreIntStatus
                    0160   endif
                    0161 ;
                    0162 ;*****************************************************************************************
                    0288
                    0289
                    0290 ;*************************************************************************************************************
                    0291
                    0292         END
                    0293
                    0294
                    0295
                    0296


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX

0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXX---------- -------- --------

All other memory blocks unused.


Errors   :     0
Warnings :     0
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN557

Four Channel Digital Voltmeter with Display and Keyboard

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

The PIC16C71 is a member of the mid-range family of 8-bit, high-speed microcontrollers, namely the PIC16CXXX. The salient features of the PIC16C71 are:

- Improved and enhanced instruction set
- 14-bit instruction word
- Interrupt capability
- On-chip four channel, 8-bit A/D converter

This application note demonstrates the capability of the PIC16C71. This application note has been broken down into four subsections:

Multiplexing Four 7-Segment LED Displays

Multiplexing Four 7-Segment LED Displays and Scanning a 4x4 Keypad

Multiplexing Four 7-Segment LED Displays and the A/D Channel0

Multiplexing Four 7-Segment LED Displays with a 4x4 Keypad and 4 A/D Channels

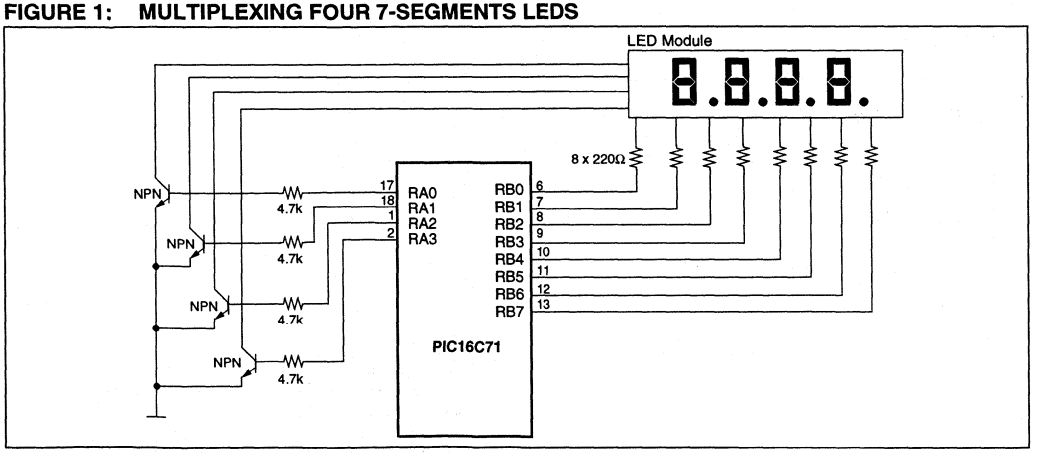
MULTIPLEXING FOUR 7-SEGMENT LED DISPLAYS

Hardware

The PIC16C71's I/O ports have an improved sink/source specification. Each I/O pin can sink up to 25 mA and source 20 mA, in addition total PORTB source current is 100 mA and sink current is 150 mA. PORTA is rated for 50 mA source current and 80 mA sink current. This makes the PIC16C71 ideal for driving 7-segment LEDs. Since the total number of I/O pins is limited to 13, the 8-bit PORTB is used to drive the 4 LEDs, while external sink transistors or MOSFETs are used to sink the digit current (Figure 1). Another alternative is to use ULN2003 open collector sink current drivers, which are available in 16-pin DIP or very small SO-16 packages. Each transistor on the ULN2003 can sink a maximum of 500 mA and the base drive can be directly driven from the PORTA pins.

3

FIGURE 1: MULTIPLEXING FOUR 7-SEGMENTS LEDS



© 1997 Microchip Technology Inc.

Software

The multiplexing is achieved by turning on each LED for a 5 µs duration every 20 µs. This gives an update rate of 50 Hz, which is quite acceptable to the human eye as a steady display. The 5 µs time-base is generated by dividing the 4.096 MHz oscillator clock. The internal prescaler is configured to be a divide by 32 and assigned to Timer0. TMR0 is pre-loaded with a value = 96. TMR0 will increment to FFh and then roll over to 00h after a period =
(256-96)•(32•4/4096000) = 5 µs.

When TMR0 rolls over, flag bit T0IF flag is set, and because bits T0IE and GIE are enabled, an interrupt is generated.

The software implements a simple timer which increments at a 1 second rate. Every second, the 4 nibble (two 8-bit registers MsdTime and LsdTime) are incremented in a BCD format. The lower 4 bits of LsdTime correspond to the least significant digit (LSD) on the display. The high 4 bits of LsdTime correspond to the second significant digit of the display and so on. Depending on which display is turned on, the corresponding 4-bit BCD value is extracted from either MsdTime or LsdTime, and decoded to a 7-segment display. The TMR0 interrupt is generated at a steady rate of 5 µs and given an instruction time of 1 µs. The entire display update program can reside in the interrupt service routine with no chance of getting an interrupt within an interrupt. The Code Listing for this section is in Appendix A.

MULTIPLEXING FOUR 7-SEGMENT LED DISPLAYS AND SCANNING A 4x4 KEYPAD

Hardware

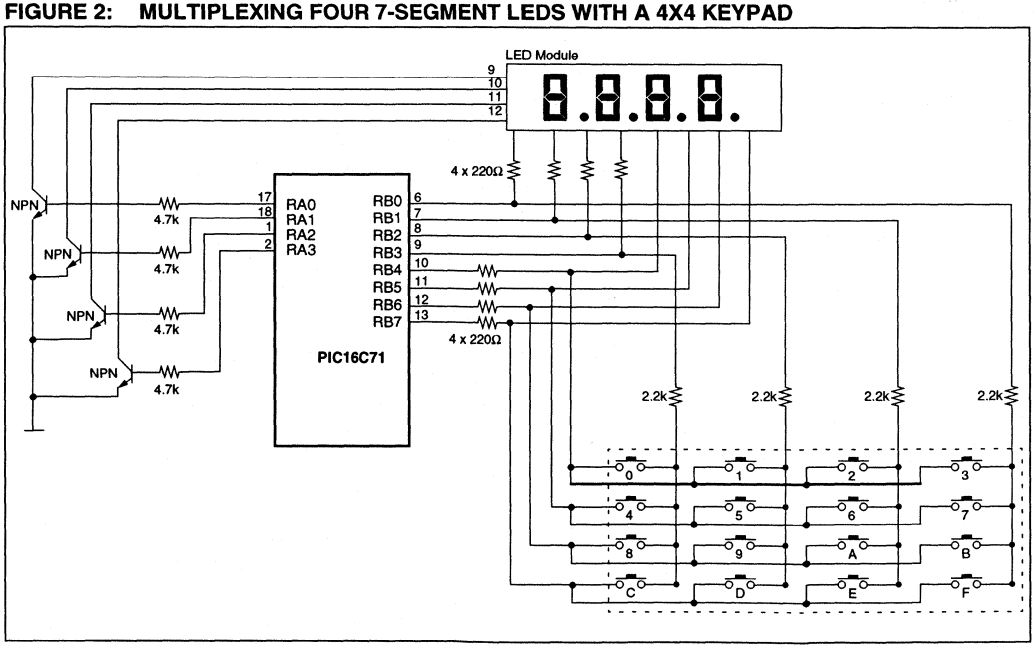
A 4x4 keypad can be very easily interfaced to the PIC16C71's PORTB (Figure 2). Internal pull-ups on pins RB7:RB4 can be enabled/disabled by clearing/setting bit $\overline{\text{RBPU}}$ (OPTION<7>). The internal pull-ups have a value of 20k at 5V (typical). In order to sense a low level at the input, the switch is "connected" to ground through a 2.2 kΩ resistor. A key hit normally lasts anywhere from 50 ms to as long as a person holds the key down. In order not to miss any key hits, the keypad is sampled every 20 µs (just after the update of the MSD).

Software

To sample the keypad, the digit sinks are first disabled. PORTB is then configured with RB7:RB4 as inputs and RB3:RB0 as outputs driven high. The pull-ups on RB7:RB4 are enabled. Sequentially RB3:RB0 are made low while RB7:RB4 are checked for a key hit (a low level). One key hit per scan is demonstrated in this program. Multiple key hits per scan can very easily be implemented. Once the key hit is sensed, a 40 ms debounce period elapses before key sampling is resumed. No more key hits are sensed until the present key is released. This prevents erroneous key inputs.

The program basically inputs the key hit and displays its value as a hexadecimal character on the multiplexed 7- segment LEDs. The Code Listing for this section is in Appendix B.

FIGURE 2: MULTIPLEXING FOUR 7-SEGMENT LEDS WITH A 4X4 KEYPAD



© 1997 Microchip Technology Inc.

MULTIPLEXING FOUR 7-SEGMENT LED DISPLAYS AND THE A/D CHANNEL0

Hardware

The four analog channels are connected to RA3:RA0. If any of these pins are used normally as digital I/O, they can momentarily be used as analog inputs. In order to avoid interference from the analog source, it is advisable to buffer the analog input through a voltage follower op-amp, however, it is not always necessary. Figure 3 and Figure 4 show some typical configurations. In this application, the analog input is a potentiometer whose wiper is connected through an RC network to Channel0. The RC is necessary in order to smooth out the analog voltage. The RC does contribute to a delay in the sampling time, however the stability of the analog reading is greatly improved.

Software

The analog input is sampled every 20 ms. The digit sinks and the drivers are turned off (i.e., PORTA is configured as an input and PORTB outputs are made low). A 1 ms settling time is allowed for the external RC network connected to the analog input to settle and then the A/D conversion is started. The result is read then converted from an 8-bit binary value to a 3-digit Binary Code Decimal (BCD) value which is then displayed on the 7-segment LEDs. The Code Listing for this section is in Appendix C.

FIGURE 3: TYPICAL CONNECTION FOR ANALOG/DIGITAL INPUT

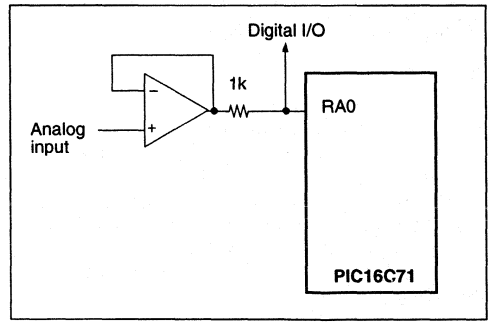
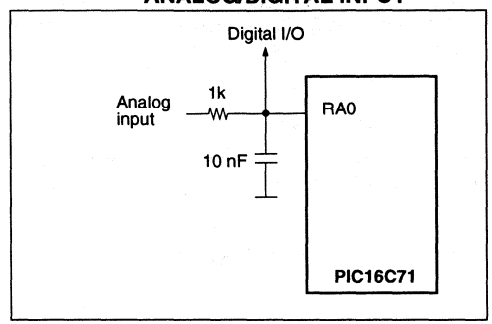


FIGURE 4: TYPICAL CONNECTION FOR ANALOG/DIGITAL INPUT



3

MULTIPLEXING FOUR 7-SEGMENT LED DISPLAYS WITH A 4x4 KEYPAD AND 4 A/D CHANNELS

Hardware

This section essentially incorporates the previous three sections to give a complete four channel voltmeter. Figure 5 shows a typical configuration. The analog channels are connected through individual potentiometers to their respective analog inputs and are sampled every 20 ms in a round robin fashion. The sampling rate can be increased to as fast as once every 5 µs if required. The keypad sampling need not be any faster than once every 20 µs.

Software

The program samples the analog inputs and saves the result in four consecutive locations starting at "ADVALUE", with Channel0 saved at the first location and so on.

KEY 0 → Channel0

or

KEY 1 → Channel0

Key hits greater than 3 are ignored. The code listing for this section is in Appendix D.

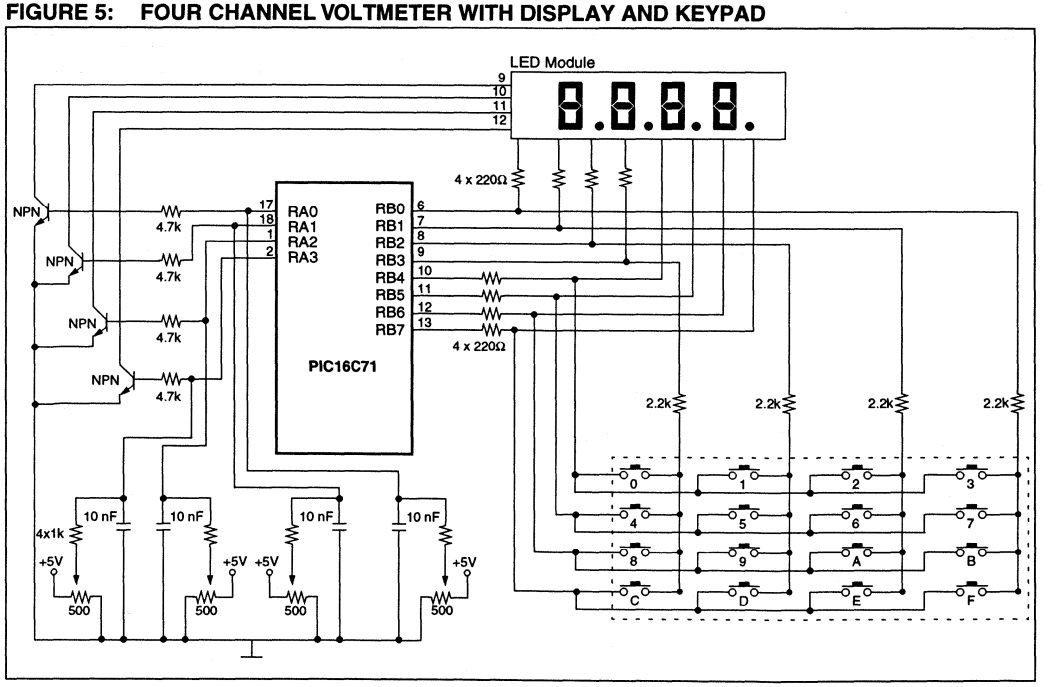
Code Size

Four 7-segment LEDs	Program Memory:	139
	Data Memory:	6
Four 7-segment LEDs and 4x4 keypad sampling	Program Memory:	207
	Data Memory:	13
Four 7-segment LEDs and A/D	Program Memory:	207
	Data Memory:	11
Four 7-segment LEDs, 4x4 keypad sampling, and A/D	Program Memory:	207
	Data Memory:	13

CONCLUSION

The four A/D channels on the PIC16C71 can be multiplexed with digital I/O, thus reducing overall pin counts and improving I/O pin usage in an analog application.

FIGURE 5: FOUR CHANNEL VOLTMETER WITH DISPLAY AND KEYPAD



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A

```
MPASM 01.40 Released            MPLX.ASM   1-16-1997  16:20:47         PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;*******************************************************************
                     00002 ;This program demonstrates how to multiplex four 7 segment LED
                     00003 ;digits using a PIC16C71. The four digits will start at 0000 and
                     00004 ;increment at a 1 sec rate up to 9999.
                     00005 ;The LEDs are updated every 5 mS, for a multiplexing rate of 20 mS.
                     00006 ;The TMR0 timer is used in internal interrupt mode to generate the
                     00007 ;5 mS.
                     00008 ;
                     00009 ;                                        Stan D'Souza 5/8/93
                     00010 ;
                     00011 ;       Program:          MPLX.ASM
                     00012 ;       Revision Date:
                     00013 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00014 ;
                     00015 ;*******************************************************************
                     00016         LIST P=16C71
                     00017         ERRORLEVEL  -302
                     00018 ;
                     00019         include   <p16c71.inc>
                     00001         LIST
                     00002 ; P16C71.INC  Standard Header File, Version 1.00  Microchip Technology
                     00142         LIST
                     00020 ;
  0000000C           00021 TempC   equ     0x0c            ;temp general purpose regs
  0000000D           00022 TempD   equ     0x0d
  0000000E           00023 TempE   equ     0x0e
  0000000F           00024 Count   equ     0x0f            ;count
  00000010           00025 MsdTime equ     0x10            ;most significant Timer
  00000011           00026 LsdTime equ     0x11            ;Least significant Timer
  00000001           00027 OptionReg equ   1
  00000002           00028 PCL     equ     2
  00000026           00029 BcdMsd  equ     26
  00000027           00030 Bcd     equ     27
                     00031 ;
0000                 00032         org     0
0000 2805            00033         goto    Start           ;skip over interrupt vector
                     00034 ;
0004                 00035         org     4
0004 281D            00036         goto    ServiceInterrupts
                     00037 ;
0005                 00038 Start
0005 2008            00039         call    InitPorts
0006 2012            00040         call    InitTimers
0007                 00041 loop
0007 2807            00042         goto    loop
                     00043 ;
0008                 00044 InitPorts
0008 1683            00045         bsf     STATUS,RP0      ;select Bank1
0009 3003            00046         movlw   3               ;make RA0-3 digital I/O
000A 0088            00047         movwf   ADCON1          ;       /
000B 0185            00048         clrf    TRISA           ;make RA0-4 outputs
000C 0186            00049         clrf    TRISB           ;make RB0-7 outputs
000D 1283            00050         bcf     STATUS,RP0      ;select Bank0
000E 0185            00051         clrf    PORTA           ;make all outputs low
000F 0186            00052         clrf    PORTB           ;       /
```

```
0010 1585          00053         bsf     PORTA,3         ;enable MSB digit sink
0011 0008          00054         return
                   00055 ;
                   00056 ;
                   00057 ;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
                   00058 ;the TMR0 will be incremented every 31.25uS. If TMR0 is preloaded
                   00059 ;with 96, it will take (256-96)*31.25uS to overflow i.e. 5mS. So the
                   00060 ;end result is that we get a TMR0 interrupt every 5mS.
0012               00061 InitTimers
0012 0190          00062         clrf    MsdTime                 ;clr timers
0013 0191          00063         clrf    LsdTime                 ;       /
0014 1683          00064         bsf     STATUS,RP0              ;select Bank1
0015 3084          00065         movlw   B'10000100'             ;assign ps to TMR0
0016 0081          00066         movwf   OptionReg               ;ps = 32
0017 1283          00067         bcf     STATUS,RP0              ;select Bank0
0018 3020          00068         movlw   B'00100000'             ;enable TMR0 interrupt
0019 008B          00069         movwf   INTCON                  ;
001A 3060          00070         movlw   .96                     ;preload TMR0
001B 0081          00071         movwf   TMR0                    ;start counter
001C 0009          00072         retfie
                   00073 ;
001D               00074 ServiceInterrupts
001D 190B          00075         btfsc   INTCON,T0IF             ;TMR0 interrupt?
001E 2822          00076         goto    ServiceTMR0             ;yes then service
001F 3020          00077         movlw   B'00100000'             ;else clr rest
0020 008B          00078         movwf   INTCON
0021 0009          00079         retfie
                   00080 ;
0022               00081 ServiceTMR0
0022 3060          00082         movlw   .96                     ;initialize TMR0
0023 0081          00083         movwf   TMR0
0024 110B          00084         bcf     INTCON,T0IF             ;clr int flag
0025 2028          00085         call    IncTimer                ;inc timer
0026 2050          00086         call    UpdateDisplay           ;update display
0027 0009          00087         retfie
                   00088 ;
                   00089 ;The display is incremented every 200*5mS = 1 Sec.
0028               00090 IncTimer
0028 0A0F          00091         incf    Count,W                 ;inc count
0029 3AC8          00092         xorlw   .200                    ;= 200?
002A 1903          00093         btfsc   STATUS,Z                ;no then skip
002B 282E          00094         goto    DoIncTime               ;else inc time
002C 0A8F          00095         incf    Count, F
002D 0008          00096         return
002E               00097 DoIncTime
002E 018F          00098         clrf    Count                   ;clr count
002F 0A11          00099         incf    LsdTime,W               ;get lsd
0030 390F          00100         andlw   0x0F                    ;mask high nibble
0031 3A0A          00101         xorlw   0x0a                    ; = 10?
0032 1903          00102         btfsc   STATUS,Z                ;no then skip
0033 2836          00103         goto    IncSecondLsd            ;inc next lsd
0034 0A91          00104         incf    LsdTime, F              ;else inc timer
0035 0008          00105         return
0036               00106 IncSecondLsd
0036 0E11          00107         swapf   LsdTime,W               ;get hi in low nibble
0037 390F          00108         andlw   0x0F                    ;mask hi nibble
0038 3E01          00109         addlw   1                       ;inc it
0039 0091          00110         movwf   LsdTime                 ;restore back
003A 0E91          00111         swapf   LsdTime, F              ;       /
003B 3A0A          00112         xorlw   0x0a                    ; = 10?
003C 1903          00113         btfsc   STATUS,Z                ;no then skip
003D 283F          00114         goto    IncThirdLsd             ;else inc next lsd
003E 0008          00115         return
003F               00116 IncThirdLsd
003F 0191          00117         clrf    LsdTime
0040 0A10          00118         incf    MsdTime,W               ;get 3rd lsd
```


© 1997 Microchip Technology Inc.

```
0041 390F           00119          andlw   0x0F                ;mask hi nibble
0042 3A0A           00120          xorlw   0x0a                ;= 10?
0043 1903           00121          btfsc   STATUS,Z            ;no then skip
0044 2847           00122          goto    IncMsd              ;else Msd
0045 0A90           00123          incf    MsdTime, F          ;else inc timer
0046 0008           00124          return
0047                00125 IncMsd
0047 0E10           00126          swapf   MsdTime,W           ;get hi in lo nibble
0048 390F           00127          andlw   0x0F                ;mask hi nibble
0049 3E01           00128          addlw   1                   ;inc timer
004A 0090           00129          movwf   MsdTime             ;restore back
004B 0E90           00130          swapf   MsdTime, F          ;       /
004C 3A0A           00131          xorlw   0x0a                ;= 10?
004D 1903           00132          btfsc   STATUS,Z            ;no then skip
004E 0190           00133          clrf    MsdTime             ;clr msd
004F 0008           00134          return
                    00135 ;
                    00136 ;
0050                00137 UpdateDisplay
0050 0805           00138          movf    PORTA,W             ;present sink value in w
0051 0185           00139          clrf    PORTA               ;disable all digits sinks
0052 390F           00140          andlw   0x0f
0053 008C           00141          movwf   TempC               ;save sink value in tempC
0054 160C           00142          bsf     TempC,4             ;preset for lsd sink
0055 0C8C           00143          rrf     TempC, F            ;determine next sink value
0056 1C03           00144          btfss   STATUS,C            ;c=1?
0057 118C           00145          bcf     TempC,3             ;no then reset LSD sink
0058 180C           00146          btfsc   TempC,0             ;else see if Msd
0059 286B           00147          goto    UpdateMsd           ;yes then do Msd
005A 188C           00148          btfsc   TempC,1             ;see if 3rdLsd
005B 2866           00149          goto    Update3rdLsd        ;yes then do 3rd Lsd
005C 190C           00150          btfsc   TempC,2             ;see if 2nd Lsd
005D 2861           00151          goto    Update2ndLsd        ;yes then do 2nd lsd
005E                00152 UpdateLsd
005E 0811           00153          movf    LsdTime,W           ;get Lsd in w
005F 390F           00154          andlw   0x0f                ;       /
0060 286F           00155          goto    DisplayOut          ;enable display
0061                00156 Update2ndLsd
0061 2080           00157          call    Chk2LsdZero         ;msd = 0 & 2 lsd 0?
0062 1D03           00158          btfss   STATUS,Z            ;yes then skip
0063 0E11           00159          swapf   LsdTime,W           ;get 2nd Lsd in w
0064 390F           00160          andlw   0x0f                ;mask rest
0065 286F           00161          goto    DisplayOut          ;enable display
0066                00162 Update3rdLsd
0066 2088           00163          call    ChkMsdZero          ;msd = 0?
0067 1D03           00164          btfss   STATUS,Z            ;yes then skip
0068 0810           00165          movf    MsdTime,W           ;get 3rd Lsd in w
0069 390F           00166          andlw   0x0f                ;mask low nibble
006A 286F           00167          goto    DisplayOut          ;enable display
006B                00168 UpdateMsd
006B 0E10           00169          swapf   MsdTime,W           ;get Msd in w
006C 390F           00170          andlw   0x0f                ;mask rest
006D 1903           00171          btfsc   STATUS,Z            ;msd != 0 then skip
006E 300A           00172          movlw   0x0a
006F                00173 DisplayOut
006F 2074           00174          call    LedTable            ;get digit output
0070 0086           00175          movwf   PORTB               ;drive leds
0071 080C           00176          movf    TempC,W             ;get sink value in w
0072 0085           00177          movwf   PORTA
0073 0008           00178          return
                    00179 ;
                    00180 ;
0074                00181 LedTable
0074 0782           00182          addwf   PCL, F               ;add to PC low
0075 343F           00183          retlw   B'00111111'          ;led drive for 0
0076 3406           00184          retlw   B'00000110'          ;led drive for 1
```


© 1997 Microchip Technology Inc.

```
0077 345B           00185         retlw   B'01011011'             ;led drive for 2
0078 344F           00186         retlw   B'01001111'             ;led drive for 3
0079 3466           00187         retlw   B'01100110'             ;led drive for 4
007A 346D           00188         retlw   B'01101101'             ;led drive for 5
007B 347D           00189         retlw   B'01111101'             ;led drive for 6
007C 3407           00190         retlw   B'00000111'             ;led drive for 7
007D 347F           00191         retlw   B'01111111'             ;led drive for 8
007E 3467           00192         retlw   B'01100111'             ;led drive for 9
007F 3400           00193         retlw   B'00000000'             ;blank led drive
                    00194 ;
                    00195 ;
0080                00196 Chk2LsdZero
0080 2088           00197         call    ChkMsdZero              ;msd = 0?
0081 1D03           00198         btfss   STATUS,Z                ;yes then skip
0082 0008           00199         return                          ;else return
0083 0E11           00200         swapf   LsdTime,W               ;get 2nd lsd
0084 390F           00201         andlw   0x0f                    ;mask of LSD
0085 1D03           00202         btfss   STATUS,Z                ;0? then skip
0086 0008           00203         return
0087 340A           00204         retlw   .10                     ;else return with 10
                    00205 ;
0088                00206 ChkMsdZero
0088 0810           00207         movf    MsdTime,W               ;get Msd in w
0089 1D03           00208         btfss   STATUS,Z                ;= 0? skip
008A 0008           00209         return                          ;else return
008B 340A           00210         retlw   .10                     ;ret with 10
                    00211 ;
                    00212
                    00213         end

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXX---- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   137
Program Memory Words Free:   887
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     3 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B

```
MPASM 01.40 Released          MPLXKEY.ASM   1-16-1997  16:24:40         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;*******************************************************************
                     00002 ;This program is to demonstrate how to multiplex four 7 segment LED
                     00003 ;digits and a 4x4 keypad using a PIC16C71.
                     00004 ;The four digits will start as '0000' and when a key is hit
                     00005 ;it is displayed on the 7 segment leds as a hex value 0 to F. The last
                     00006 ;digit hit is always displayed on the right most led with the rest of
                     00007 ;the digits shifted to the left. The left most digit is deleted.
                     00008 ;The LEDs are updated every 20mS, the keypad is scanned at a rate of 20
                     00009 ;mS. The TMR0 timer is used in internal interrupt mode to generate the
                     00010 ;5 mS.
                     00011 ;
                     00012 ;                                        Stan D'Souza 5/8/93
                     00013 ;
                     00014 ;       Program:          MPLXKEY.ASM
                     00015 ;       Revision Date:
                     00016 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00017 ;
                     00018 ;*******************************************************************
                     00019         LIST P=16C71
                     00020         ERRORLEVEL  -302
                     00021 ;
                     00022         include     <p16c71.inc>
                     00001         LIST
                     00002 ; P16C71.INC Standard Header File,Ver. 1.00 Microchip Technology, Inc.
                     00142         LIST
                     00023 ;
0000000C             00024 TempC   equ     0x0c            ;temp general purpose regs
0000000D             00025 TempD   equ     0x0d
0000000E             00026 TempE   equ     0x0e
00000020             00027 PABuf   equ     0x20
00000021             00028 PBBuf   equ     0x21
0000000F             00029 Count   equ     0x0f            ;count
00000010             00030 MsdTime equ     0x10            ;most significant Timer
00000011             00031 LsdTime equ     0x11            ;Least significant Timer
00000012             00032 KeyFlag equ     0x12            ;flags related to key pad
00000000             00033 keyhit  equ     0               ;bit 0 --> key-press on
00000001             00034 DebnceOn equ    1               ;bit 1 --> debounce on
00000002             00035 noentry equ     2               ;no key entry = 0
00000003             00036 ServKey equ     3               ;bit 3 --> service key
00000013             00037 Debnce  equ     0x13            ;debounce counter
00000014             00038 NewKey  equ     0x14
0000002F             00039 WBuffer equ     0x2f
0000002E             00040 StatBuffer equ  0x2e
00000001             00041 OptionReg equ   1
00000002             00042 PCL     equ     2
                     00043 ;
                     00044 ;
                     00045 push    macro
                     00046         movwf   WBuffer         ;save w reg in Buffer
                     00047         swapf   WBuffer, F      ;swap it
                     00048         swapf   STATUS,W        ;get status
                     00049         movwf   StatBuffer      ;save it
                     00050         endm
                     00051 ;
```

```
                    00052 pop     macro
                    00053         swapf   StatBuffer,W    ;restore status
                    00054         movwf   STATUS          ;       /
                    00055         swapf   WBuffer,W       ;restore W reg
                    00056         endm
                    00057 ;
0000                00058         org     0
0000 280D           00059         goto    Start           ;skip over interrupt vector
                    00060 ;
0004                00061         org     4
                    00062 ;It is always a good practice to save and restore the w reg,
                    00063 ;and the status reg during an interrupt.
                    00064         push
0004 00AF               M         movwf   WBuffer         ;save w reg in Buffer
0005 0EAF               M         swapf   WBuffer, F      ;swap it
0006 0E03               M         swapf   STATUS,W        ;get status
0007 00AE               M         movwf   StatBuffer      ;save it
0008 2036           00065         call    ServiceInterrupts
                    00066         pop
0009 0E2E               M         swapf   StatBuffer,W    ;restore status
000A 0083               M         movwf   STATUS          ;       /
000B 0E2F               M         swapf   WBuffer,W       ;restore W reg
000C 0009           00067         retfie
                    00068 ;
000D                00069 Start
000D 2020           00070         call    InitPorts
000E 202A           00071         call    InitTimers
000F                00072 loop
000F 1992           00073         btfsc   KeyFlag,ServKey ;key service pending
0010 2012           00074         call    ServiceKey      ;yes then service
0011 280F           00075         goto    loop
                    00076 ;
                    00077 ;ServiceKey, does the software service for a keyhit. After a key
                    00078 ;service, the ServKey flag is reset, to denote a completed operation.
0012                00079 ServiceKey
0012 0814           00080         movf    NewKey,W        ;get key value
0013 008E           00081         movwf   TempE           ;save in TempE
0014 0E10           00082         swapf   MsdTime,W       ;move MSD out
0015 39F0           00083         andlw   B'11110000'     ;clr lo nibble
0016 0090           00084         movwf   MsdTime         ;save back
0017 0E11           00085         swapf   LsdTime,W       ;get Lsd
0018 390F           00086         andlw   B'00001111'     ;mask off lsd
0019 0490           00087         iorwf   MsdTime, F      ;and left shift 3rd
001A 0E11           00088         swapf   LsdTime,W       ;get Lsd again
001B 39F0           00089         andlw   B'11110000'     ;mask off 2nd
001C 040E           00090         iorwf   TempE,W         ;or with new lsd
001D 0091           00091         movwf   LsdTime         ;make Lsd
001E 1192           00092         bcf     KeyFlag,ServKey ;reset service flag
001F 0008           00093         return
                    00094
                    00095 ;
0020                00096 InitPorts
0020 1683           00097         bsf     STATUS,RP0      ;select Bank1
0021 3003           00098         movlw   3               ;make RA0-3 digital I/O
0022 0088           00099         movwf   ADCON1          ;       /
0023 0185           00100         clrf    TRISA           ;make RA0-4 outputs
0024 0186           00101         clrf    TRISB           ;make RB0-7 outputs
0025 1283           00102         bcf     STATUS,RP0      ;select Bank0
0026 0185           00103         clrf    PORTA           ;make all outputs low
0027 0186           00104         clrf    PORTB           ;       /
0028 1585           00105         bsf     PORTA,3         ;enable MSB digit sink
0029 0008           00106         return
                    00107 ;
                    00108 ;
                    00109 ;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
                    00110 ;the TMR0 will be incremented every 31.25uS. If TMR0 is preloaded
```


© 1997 Microchip Technology Inc.

```
                00111 ;with 96, it will take (256-96)*31.25uS to overflow i.e. 5mS. So the
                00112 ;end result is that we get a TMR0 interrupt every 5mS.
002A            00113 InitTimers
002A 0190       00114         clrf    MsdTime         ;clr timers
002B 0191       00115         clrf    LsdTime         ;       /
002C 0192       00116         clrf    KeyFlag         ;clr all flags
002D 1683       00117         bsf     STATUS,RP0      ;select Bank1
002E 3084       00118         movlw   B'10000100'     ;assign ps to TMR0
002F 0081       00119         movwf   OptionReg       ;ps = 32
0030 1283       00120         bcf     STATUS,RP0      ;select Bank0
0031 3020       00121         movlw   B'00100000'     ;enable TMR0 interrupt
0032 008B       00122         movwf   INTCON          ;
0033 3060       00123         movlw   .96             ;preload TMR0
0034 0081       00124         movwf   TMR0            ;start counter
0035 0009       00125         retfie
                00126 ;
0036            00127 ServiceInterrupts
0036 190B       00128         btfsc   INTCON,T0IF     ;TMR0 interrupt?
0037 283B       00129         goto    ServiceTMR0     ;yes then service
0038 018B       00130         clrf    INTCON          ;else clr all int
0039 168B       00131         bsf     INTCON,T0IE
003A 0008       00132         return
                00133 ;
003B            00134 ServiceTMR0
003B 3060       00135         movlw   .96             ;initialize TMR0
003C 0081       00136         movwf   TMR0
003D 110B       00137         bcf     INTCON,T0IF     ;clr int flag
003E 1805       00138         btfsc   PORTA,0         ;if msb on then do
003F 2042       00139         call    ScanKeys        ;do a quick key scan
0040 20A1       00140         call    UpdateDisplay   ;update display
0041 0008       00141         return
                00142 ;
                00143 ;
                00144 ;ScanKeys, scans the 4X4 keypad matrix and returns a key value in
                00145 ;NewKey (0 - F) if a key is pressed, if not it clears the keyhit flag.
                00146 ;Debounce for a given keyhit is also taken care of.
                00147 ;The rate of key scan is 20mS with a 4.096Mhz clock.
0042            00148 ScanKeys
0042 1C92       00149         btfss   KeyFlag,DebnceOn ;debounce on?
0043 2848       00150         goto    Scan1            ;no then scan keypad
0044 0B93       00151         decfsz  Debnce, F        ;else dec debounce time
0045 0008       00152         return                   ;not over then return
0046 1092       00153         bcf     KeyFlag,DebnceOn ;over, clr debounce flag
0047 0008       00154         return                   ;and return
0048            00155 Scan1
0048 208A       00156         call    SavePorts        ;save port values
0049 30EF       00157         movlw   B'11101111'      ;init TempD
004A 008D       00158         movwf   TempD
004B            00159 ScanNext
004B 0806       00160         movf    PORTB,W          ;read to init port
004C 100B       00161         bcf     INTCON,RBIF      ;clr flag
004D 0C8D       00162         rrf     TempD, F         ;get correct column
004E 1C03       00163         btfss   STATUS,C         ;if carry set?
004F 2862       00164         goto    NoKey            ;no then end
0050 080D       00165         movf    TempD,W          ;else output
0051 0086       00166         movwf   PORTB            ;low column scan line
0052 0000       00167         nop
0053 1C0B       00168         btfss   INTCON,RBIF      ;flag set?
0054 284B       00169         goto    ScanNext         ;no then next
0055 1812       00170         btfsc   KeyFlag,keyhit   ;last key released?
0056 2860       00171         goto    SKreturn         ;no then exit
0057 1412       00172         bsf     KeyFlag,keyhit   ;set new key hit
0058 0E06       00173         swapf   PORTB,W          ;read port
0059 008E       00174         movwf   TempE            ;save in TempE
005A 2064       00175         call    GetKeyValue      ;get key value 0 - F
005B 0094       00176         movwf   NewKey           ;save as New key
```

```
005C 1592          00177         bsf     KeyFlag,ServKey  ;set service flag
005D 1492          00178         bsf     KeyFlag,DebnceOn ;set flag
005E 3004          00179         movlw   4
005F 0093          00180         movwf   Debnce           ;load debounce time
0060               00181 SKreturn
0060 2097          00182         call    RestorePorts     ;restore ports
0061 0008          00183         return
                   00184 ;
0062               00185 NoKey
0062 1012          00186         bcf     KeyFlag,keyhit   ;clr flag
0063 2860          00187         goto    SKreturn
                   00188 ;
                   00189 ;GetKeyValue gets the key as per the following layout
                   00190 ;
                   00191 ;                  Col1    Col2    Col3    Col4
                   00192 ;                  (RB3)   (RB2)   (RB1)   (RB0)
                   00193 ;
                   00194 ;Row1(RB4)           0       1       2       3
                   00195 ;
                   00196 ;Row2(RB5)           4       5       6       7
                   00197 ;
                   00198 ;Row3(RB6)           8       9       A       B
                   00199 ;
                   00200 ;Row4(RB7)           C       D       E       F
                   00201 ;
0064               00202 GetKeyValue
0064 018C          00203         clrf    TempC
0065 1D8D          00204         btfss   TempD,3          ;first column
0066 286E          00205         goto    RowValEnd
0067 0A8C          00206         incf    TempC, F
0068 1D0D          00207         btfss   TempD,2          ;second col.
0069 286E          00208         goto    RowValEnd
006A 0A8C          00209         incf    TempC, F
006B 1C8D          00210         btfss   TempD,1          ;3rd col.
006C 286E          00211         goto    RowValEnd
006D 0A8C          00212         incf    TempC, F         ;last col.
006E               00213 RowValEnd
006E 1C0E          00214         btfss   TempE,0          ;top row?
006F 2878          00215         goto    GetValCom        ;yes then get 0,1,2&3
0070 1C8E          00216         btfss   TempE,1          ;2nd row?
0071 2877          00217         goto    Get4567          ;yes the get 4,5,6&7
0072 1D0E          00218         btfss   TempE,2          ;3rd row?
0073 2875          00219         goto    Get89ab          ;yes then get 8,9,a&b
0074               00220 Getcdef
0074 150C          00221         bsf     TempC,2          ;set msb bits
0075               00222 Get89ab
0075 158C          00223         bsf     TempC,3          ;       /
0076 2878          00224         goto    GetValCom        ;do common part
0077               00225 Get4567
0077 150C          00226         bsf     TempC,2
0078               00227 GetValCom
0078 080C          00228         movf    TempC,W
0079 0782          00229         addwf   PCL, F
007A 3400          00230         retlw   0
007B 3401          00231         retlw   1
007C 3402          00232         retlw   2
007D 3403          00233         retlw   3
007E 3404          00234         retlw   4
007F 3405          00235         retlw   5
0080 3406          00236         retlw   6
0081 3407          00237         retlw   7
0082 3408          00238         retlw   8
0083 3409          00239         retlw   9
0084 340A          00240         retlw   0a
0085 340B          00241         retlw   0b
0086 340C          00242         retlw   0c
```


© 1997 Microchip Technology Inc.

```
0087 340D           00243          retlw   0d
0088 340E           00244          retlw   0e
0089 340F           00245          retlw   0f
                    00246 ;
                    00247 ;SavePorts, saves the porta and portb condition during a key scan
                    00248 ;operation.
008A                00249 SavePorts
008A 0805           00250          movf    PORTA,W         ;Get sink value
008B 00A0           00251          movwf   PABuf           ;save in buffer
008C 0185           00252          clrf    PORTA           ;disable all sinks
008D 0806           00253          movf    PORTB,W         ;get port b
008E 00A1           00254          movwf   PBBuf           ;save in buffer
008F 30FF           00255          movlw   0xff            ;make all high
0090 0086           00256          movwf   PORTB           ;on port b
0091 1683           00257          bsf     STATUS,RP0      ;select Bank1
0092 1381           00258          bcf     OptionReg,7     ;enable pull ups
0093 30F0           00259          movlw   B'11110000'     ;port b hi nibble inputs
0094 0086           00260          movwf   TRISB           ;lo nibble outputs
0095 1283           00261          bcf     STATUS,RP0      ;Bank0
0096 0008           00262          return
                    00263 ;
                    00264 ;RestorePorts, restores the condition of porta and portb after a
                    00265 ;key scan operation.
0097                00266 RestorePorts
0097 0821           00267          movf    PBBuf,W         ;get port b
0098 0086           00268          movwf   PORTB
0099 0820           00269          movf    PABuf,W         ;get port a value
009A 0085           00270          movwf   PORTA
009B 1683           00271          bsf     STATUS,RP0      ;select Bank1
009C 1781           00272          bsf     OptionReg,7     ;disable pull ups
009D 0185           00273          clrf    TRISA           ;make port a outputs
009E 0186           00274          clrf    TRISB           ;as well as PORTB
009F 1283           00275          bcf     STATUS,RP0      ;Bank0
00A0 0008           00276          return
                    00277 ;
                    00278 ;
00A1                00279 UpdateDisplay
00A1 0805           00280          movf    PORTA,W         ;present sink value in w
00A2 0185           00281          clrf    PORTA           ;disable all digits sinks
00A3 390F           00282          andlw   0x0f
00A4 008C           00283          movwf   TempC           ;save sink value in tempC
00A5 160C           00284          bsf     TempC,4         ;preset for lsd sink
00A6 0C8C           00285          rrf     TempC, F        ;determine next sink value
00A7 1C03           00286          btfss   STATUS,C        ;c=1?
00A8 118C           00287          bcf     TempC,3         ;no then reset LSD sink
00A9 180C           00288          btfsc   TempC,0         ;else see if Msd
00AA 28B8           00289          goto    UpdateMsd       ;yes then do Msd
00AB 188C           00290          btfsc   TempC,1         ;see if 3rdLsd
00AC 28B5           00291          goto    Update3rdLsd    ;yes then do 3rd Lsd
00AD 190C           00292          btfsc   TempC,2         ;see if 2nd Lsd
00AE 28B2           00293          goto    Update2ndLsd    ;yes then do 2nd lsd
00AF                00294 UpdateLsd
00AF 0811           00295          movf    LsdTime,W       ;get Lsd in w
00B0 390F           00296          andlw   0x0f            ;       /
00B1 28BA           00297          goto    DisplayOut
00B2                00298 Update2ndLsd
00B2 0E11           00299          swapf   LsdTime,W       ;get 2nd Lsd in w
00B3 390F           00300          andlw   0x0f            ;mask rest
00B4 28BA           00301          goto    DisplayOut      ;enable display
00B5                00302 Update3rdLsd
00B5 0810           00303          movf    MsdTime,W       ;get 3rd Lsd in w
00B6 390F           00304          andlw   0x0f            ;mask low nibble
00B7 28BA           00305          goto    DisplayOut      ;enable display
00B8                00306 UpdateMsd
00B8 0E10           00307          swapf   MsdTime,W       ;get Msd in w
00B9 390F           00308          andlw   0x0f            ;mask rest
```

```
00BA                00309 DisplayOut
00BA 20BF           00310         call    LedTable        ;get digit output
00BB 0086           00311         movwf   PORTB           ;drive leds
00BC 080C           00312         movf    TempC,W         ;get sink value in w
00BD 0085           00313         movwf   PORTA
00BE 0008           00314         return
                    00315 ;
                    00316 ;
00BF                00317 LedTable
00BF 0782           00318         addwf   PCL, F          ;add to PC low
00C0 343F           00319         retlw   B'00111111'     ;led drive for 0
00C1 3406           00320         retlw   B'00000110'     ;led drive for 1
00C2 345B           00321         retlw   B'01011011'     ;led drive for 2
00C3 344F           00322         retlw   B'01001111'     ;led drive for 3
00C4 3466           00323         retlw   B'01100110'     ;led drive for 4
00C5 346D           00324         retlw   B'01101101'     ;led drive for 5
00C6 347D           00325         retlw   B'01111101'     ;led drive for 6
00C7 3407           00326         retlw   B'00000111'     ;led drive for 7
00C8 347F           00327         retlw   B'01111111'     ;led drive for 8
00C9 3467           00328         retlw   B'01100111'     ;led drive for 9
00CA 3477           00329         retlw   B'01110111'     ;led drive for A
00CB 347C           00330         retlw   B'01111100'     ;led drive for b
00CC 3439           00331         retlw   B'00111001'     ;led drive for C
00CD 345E           00332         retlw   B'01011110'     ;led drive for d
00CE 3479           00333         retlw   B'01111001'     ;led drive for E
00CF 3471           00334         retlw   B'01110001'     ;led drive for F
                    00335
                    00336                                 ;
                    00337                                 ;
                    00338
                    00339         end

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   205
Program Memory Words Free:   819


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     6 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C

```
MPASM 01.40 Released          MPLXCH0.ASM   1-16-1997  16:24:14         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;*******************************************************************
                     00002 ;This program is to demonstrate how to multiplex four 7 segment LED
                     00003 ;and sample ch0 of the a/d in a PIC16C71. The a/d value is displayed
                     00004 ;as a 3 digit decimal value of the a/d input (0 - 255).
                     00005 ;The LEDs are updated every 20mS, the a/d is sampled every 20 mS.
                     00006 ;The TIMER0 timer is used in internal interrupt mode to generate the
                     00007 ;5 mS.
                     00008 ;
                     00009 ;                                        Stan D'Souza 5/8/93
                     00010 ;
                     00011 ;
                     00012 ;
                     00013 ;       Program:          MPLXCH0.ASM
                     00014 ;       Revision Date:
                     00015 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00016 ;
                     00017 ;*******************************************************************
                     00018         LIST P=16C71
                     00019         ERRORLEVEL  -302
                     00020 ;
                     00021         include     <p16c71.inc>
                     00001         LIST
                     00002 ; P16C71.INC Standard Header File, Ver. 1.00 Microchip Technology, Inc.
                     00142         LIST
                     00022 ;
00000026             00023 BcdMsd  equ     26
00000027             00024 Bcd     equ     27
0000000C             00025 TempC   equ     0x0c            ;temp general purpose regs
0000000D             00026 TempD   equ     0x0d
0000000E             00027 TempE   equ     0x0e
00000020             00028 PABuf   equ     0x20
00000021             00029 PBBuf   equ     0x21
0000000F             00030 Count   equ     0x0f            ;count
00000010             00031 MsdTime equ     0x10            ;most significant Timer
00000011             00032 LsdTime equ     0x11            ;Least significant Timer
00000012             00033 ADFlag  equ     0x12            ;flags related to key pad
00000005             00034 ADOver  equ     5               ;bit 5 --> a/d over
0000002F             00035 WBuffer equ     0x2f
0000002E             00036 StatBuffer equ  0x2e
00000001             00037 OptionReg equ   1
00000002             00038 PCL     equ     2
                     00039 ;
                     00040 push    macro
                     00041         movwf   WBuffer         ;save w reg in Buffer
                     00042         swapf   WBuffer, F      ;swap it
                     00043         swapf   STATUS,W        ;get status
                     00044         movwf   StatBuffer      ;save it
                     00045         endm
                     00046 ;
                     00047 pop     macro
                     00048         swapf   StatBuffer,W    ;restore status
                     00049         movwf   STATUS          ;       /
                     00050         swapf   WBuffer,W       ;restore W reg
                     00051         endm
```

```
                    00052 ;
0000                00053         org     0
0000 280D           00054         goto    Start           ;skip over interrupt vector
                    00055 ;
0004                00056         org     4
                    00057 ;It is always a good practice to save and restore the w reg,
                    00058 ;and the status reg during an interrupt.
                    00059         push
0004 00AF               M         movwf   WBuffer         ;save w reg in Buffer
0005 0EAF               M         swapf   WBuffer, F      ;swap it
0006 0E03               M         swapf   STATUS,W        ;get status
0007 00AE               M         movwf   StatBuffer      ;save it
0008 2039           00060         call    ServiceInterrupts
                    00061         pop
0009 0E2E               M         swapf   StatBuffer,W    ;restore status
000A 0083               M         movwf   STATUS          ;       /
000B 0E2F               M         swapf   WBuffer,W       ;restore W reg
000C 0009           00062         retfie
                    00063 ;
000D                00064 Start
000D 2021           00065         call    InitPorts
000E 202B           00066         call    InitTimers
000F 2036           00067         call    InitAd
0010                00068 loop
0010 1A92           00069         btfsc   ADFlag,ADOver   ;a/d over?
0011 2013           00070         call    UpdateAd        ;yes then update
0012 2810           00071         goto    loop
                    00072 ;
0013                00073 UpdateAd
0013 1C88           00074         btfss   ADCON0,ADIF     ;a/d done?
0014 0008           00075         return                  ;no then leave
0015 0809           00076         movf    ADRES,W         ;get a/d value
0016 00A1           00077         movwf   L_byte
0017 01A0           00078         clrf    H_byte
0018 20AD           00079         call    B2_BCD
0019 0824           00080         movf    R2,W            ;get LSd
001A 0091           00081         movwf   LsdTime         ;save in LSD
001B 0823           00082         movf    R1,W            ;get Msd
001C 0090           00083         movwf   MsdTime         ;save in Msd
001D 1088           00084         bcf     ADCON0,ADIF     ;clr interrupt flag
001E 1008           00085         bcf     ADCON0,ADON     ;turn off a/d
001F 1292           00086         bcf     ADFlag,ADOver   ;clr flag
0020 0008           00087         return
                    00088 ;
                    00089 ;
                    00090 ;
0021                00091 InitPorts
0021 1683           00092         bsf     STATUS,RP0      ;select Bank1
0022 3003           00093         movlw   3               ;make RA0-3 digital I/O
0023 0088           00094         movwf   ADCON1          ;       /
0024 0185           00095         clrf    TRISA           ;make RA0-4 outputs
0025 0186           00096         clrf    TRISB           ;make RB0-7 outputs
0026 1283           00097         bcf     STATUS,RP0      ;select Bank0
0027 0185           00098         clrf    PORTA           ;make all outputs low
0028 0186           00099         clrf    PORTB           ;       /
0029 1585           00100         bsf     PORTA,3         ;enable MSB digit sink
002A 0008           00101         return
                    00102 ;
                    00103 ;
                    00104 ;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
                    00105 ;the TMR0 will be incremented every 31.25uS. If TMR0 is preloaded
                    00106 ;with 96, it will take (256-96)*31.25uS to overflow i.e. 5mS. So the
                    00107 ;end result is that we get a TMR0 interrupt every 5mS.
002B                00108 InitTimers
002B 0190           00109         clrf    MsdTime         ;clr timers
002C 0191           00110         clrf    LsdTime         ;       /
```


© 1997 Microchip Technology Inc.

```
002D 1683           00111         bsf     STATUS,RP0      ;select Bank1
002E 3084           00112         movlw   B'10000100'     ;assign ps to TMR0
002F 0081           00113         movwf   OptionReg       ;ps = 32
0030 1283           00114         bcf     STATUS,RP0      ;select Bank0
0031 3020           00115         movlw   B'00100000'     ;enable TMR0 interrupt
0032 008B           00116         movwf   INTCON          ;
0033 3060           00117         movlw   .96             ;preload TMR0
0034 0081           00118         movwf   TMR0            ;start counter
0035 0009           00119         retfie
                    00120 ;
                    00121 ;
0036                00122 InitAd
0036 30C0           00123         movlw   B'11000000'     ;rc osc, ch 0 for a/d
0037 0088           00124         movwf   ADCON0
0038 0008           00125         return
                    00126 ;
                    00127 ;
0039                00128 ServiceInterrupts
0039 190B           00129         btfsc   INTCON,T0IF     ;TMR0 interrupt?
003A 283E           00130         goto    ServiceTMR0     ;yes then service
003B 018B           00131         clrf    INTCON
003C 168B           00132         bsf     INTCON,T0IE
003D 0008           00133         return
                    00134 ;
003E                00135 ServiceTMR0
003E 3060           00136         movlw   .96             ;initialize TMR0
003F 0081           00137         movwf   TMR0
0040 110B           00138         bcf     INTCON,T0IF     ;clr int flag
0041 1C05           00139         btfss   PORTA,0         ;last digit?
0042 2045           00140         call    SampleAd        ;then sample a/d
0043 2071           00141         call    UpdateDisplay   ;else update display
0044 0008           00142         return
                    00143 ;
                    00144 ;
0045                00145 SampleAd
0045 205A           00146         call    SavePorts
0046 204C           00147         call    DoAd            ;do a ad conversion
0047                00148 AdDone
0047 1908           00149         btfsc   ADCON0,GO       ;ad done?
0048 2847           00150         goto    AdDone          ;no then loop
0049 1692           00151         bsf     ADFlag,ADOver   ;set a/d over flag
004A 2067           00152         call    RestorePorts    ;restore ports
004B 0008           00153         return
                    00154 ;
                    00155 ;
004C                00156 DoAd
004C 0186           00157         clrf    PORTB           ;turn off leds
004D 1683           00158         bsf     STATUS,RP0      ;select Bank1
004E 300F           00159         movlw   0x0f            ;make port a hi-Z
004F 0085           00160         movwf   TRISA           ;       /
0050 1283           00161         bcf     STATUS,RP0      ;select Bank0
0051 1408           00162         bsf     ADCON0,ADON     ;start a/d
0052 307D           00163         movlw   .125
0053 2056           00164         call    Wait
0054 1508           00165         bsf     ADCON0,GO       ;start conversion
0055 0008           00166         return
                    00167 ;
                    00168 ;
0056                00169 Wait
0056 008C           00170         movwf   TempC           ;store in temp
0057                00171 Next
0057 0B8C           00172         decfsz  TempC, F
0058 2857           00173         goto    Next
0059 0008           00174         return
                    00175
                    00176 ;
```


© 1997 Microchip Technology Inc.

```
                    00177 ;SavePorts, saves the porta and portb condition during a key scan
                    00178 ;operation.
005A                00179 SavePorts
005A 0805           00180         movf    PORTA,W         ;Get sink value
005B 00A0           00181         movwf   PABuf           ;save in buffer
005C 0185           00182         clrf    PORTA           ;disable all sinks
005D 0806           00183         movf    PORTB,W         ;get port b
005E 00A1           00184         movwf   PBBuf           ;save in buffer
005F 30FF           00185         movlw   0xff            ;make all high
0060 0086           00186         movwf   PORTB           ;on port b
0061 1683           00187         bsf     STATUS,RP0      ;select Bank1
0062 1381           00188         bcf     OptionReg,7     ;enable pull ups
0063 30F0           00189         movlw   B'11110000'     ;port b hi nibble inputs
0064 0086           00190         movwf   TRISB           ;lo nibble outputs
0065 1283           00191         bcf     STATUS,RP0      ;Bank0
0066 0008           00192         return
                    00193 ;
                    00194 ;RestorePorts, restores the condition of porta and portb after a
                    00195 ;key scan operation.
0067                00196 RestorePorts
0067 0821           00197         movf    PBBuf,W         ;get port n
0068 0086           00198         movwf   PORTB
0069 0820           00199         movf    PABuf,W         ;get port a value
006A 0085           00200         movwf   PORTA
006B 1683           00201         bsf     STATUS,RP0      ;select Bank1
006C 1781           00202         bsf     OptionReg,7     ;disable pull ups
006D 0185           00203         clrf    TRISA           ;make port a outputs
006E 0186           00204         clrf    TRISB           ;as well as PORTB
006F 1283           00205         bcf     STATUS,RP0      ;Bank0
0070 0008           00206         return
                    00207 ;
                    00208 ;
0071                00209 UpdateDisplay
0071 0805           00210         movf    PORTA,W         ;present sink value in w
0072 0185           00211         clrf    PORTA           ;disable all digits sinks
0073 390F           00212         andlw   0x0f
0074 008C           00213         movwf   TempC           ;save sink value in tempC
0075 160C           00214         bsf     TempC,4         ;preset for lsd sink
0076 0C8C           00215         rrf     TempC, F        ;determine next sink value
0077 1C03           00216         btfss   STATUS,C        ;c=1?
0078 118C           00217         bcf     TempC,3         ;no then reset LSD sink
0079 180C           00218         btfsc   TempC,0         ;else see if Msd
007A 288C           00219         goto    UpdateMsd       ;yes then do Msd
007B 188C           00220         btfsc   TempC,1         ;see if 3rdLsd
007C 2887           00221         goto    Update3rdLsd    ;yes then do 3rd Lsd
007D 190C           00222         btfsc   TempC,2         ;see if 2nd Lsd
007E 2882           00223         goto    Update2ndLsd    ;yes then do 2nd lsd
007F                00224 UpdateLsd
007F 0811           00225         movf    LsdTime,W       ;get Lsd in w
0080 390F           00226         andlw   0x0f            ;       /
0081 2890           00227         goto    DisplayOut      ;enable display
0082                00228 Update2ndLsd
0082 20A1           00229         call    Chk2LsdZero     ;msd = 0 & 2 lsd 0?
0083 1D03           00230         btfss   STATUS,Z        ;yes then skip
0084 0E11           00231         swapf   LsdTime,W       ;get 2nd Lsd in w
0085 390F           00232         andlw   0x0f            ;mask rest
0086 2890           00233         goto    DisplayOut      ;enable display
0087                00234 Update3rdLsd
0087 20A9           00235         call    ChkMsdZero      ;msd = 0?
0088 1D03           00236         btfss   STATUS,Z        ;yes then skip
0089 0810           00237         movf    MsdTime,W       ;get 3rd Lsd in w
008A 390F           00238         andlw   0x0f            ;mask low nibble
008B 2890           00239         goto    DisplayOut      ;enable display
008C                00240 UpdateMsd
008C 0E10           00241         swapf   MsdTime,W       ;get Msd in w
008D 390F           00242         andlw   0x0f            ;mask rest
```


© 1997 Microchip Technology Inc.

```
008E 1903           00243          btfsc   STATUS,Z        ;msd != 0 then skip
008F 300A           00244          movlw   0x0a
0090                00245 DisplayOut
0090 2095           00246          call    LedTable        ;get digit output
0091 0086           00247          movwf   PORTB           ;drive leds
0092 080C           00248          movf    TempC,W         ;get sink value in w
0093 0085           00249          movwf   PORTA
0094 0008           00250          return
                    00251 ;
                    00252 ;
0095                00253 LedTable
0095 0782           00254          addwf   PCL, F          ;add to PC low
0096 343F           00255          retlw   B'00111111'     ;led drive for 0
0097 3406           00256          retlw   B'00000110'     ;led drive for 1
0098 345B           00257          retlw   B'01011011'     ;led drive for 2
0099 344F           00258          retlw   B'01001111'     ;led drive for 3
009A 3466           00259          retlw   B'01100110'     ;led drive for 4
009B 346D           00260          retlw   B'01101101'     ;led drive for 5
009C 347D           00261          retlw   B'01111101'     ;led drive for 6
009D 3407           00262          retlw   B'00000111'     ;led drive for 7
009E 347F           00263          retlw   B'01111111'     ;led drive for 8
009F 3467           00264          retlw   B'01100111'     ;led drive for 9
00A0 3400           00265          retlw   B'00000000'     ;blank led drive
                    00266 ;
                    00267 ;
00A1                00268 Chk2LsdZero
00A1 20A9           00269          call    ChkMsdZero      ;msd = 0?
00A2 1D03           00270          btfss   STATUS,Z        ;yes then skip
00A3 0008           00271          return                  ;else return
00A4 0E11           00272          swapf   LsdTime,W       ;get 2nd lsd
00A5 390F           00273          andlw   0x0f            ;mask of LSD
00A6 1D03           00274          btfss   STATUS,Z        ;0? then skip
00A7 0008           00275          return
00A8 340A           00276          retlw   .10             ;else return with 10
                    00277 ;
00A9                00278 ChkMsdZero
00A9 0810           00279          movf    MsdTime,W       ;get Msd in w
00AA 1D03           00280          btfss   STATUS,Z        ;= 0? skip
00AB 0008           00281          return                  ;else return
00AC 340A           00282          retlw   .10             ;ret with 10
                    00283 ;
                    00284 ;
                    00285 ;
  00000026          00286 count  equ     26
  00000027          00287 temp   equ     27
                    00288 ;
  00000020          00289 H_byte  equ    20
  00000021          00290 L_byte  equ    21
  00000022          00291 R0      equ    22              ; RAM Assignments
  00000023          00292 R1      equ    23
  00000024          00293 R2      equ    24
                    00294 ;
                    00295 ;
00AD 1003           00296 B2_BCD  bcf    STATUS,0        ; clear the carry bit
00AE 3010           00297          movlw   .16
00AF 00A6           00298          movwf   count
00B0 01A2           00299          clrf    R0
00B1 01A3           00300          clrf    R1
00B2 01A4           00301          clrf    R2
00B3 0DA1           00302 loop16   rlf     L_byte, F
00B4 0DA0           00303          rlf     H_byte, F
00B5 0DA4           00304          rlf     R2, F
00B6 0DA3           00305          rlf     R1, F
00B7 0DA2           00306          rlf     R0, F
                    00307 ;
00B8 0BA6           00308          decfsz  count, F
```

```
00B9 28BB           00309         goto    adjDEC
00BA 3400           00310         RETLW   0
                    00311 ;
00BB 3024           00312 adjDEC  movlw   R2
00BC 0084           00313         movwf   FSR
00BD 20C5           00314         call    adjBCD
                    00315 ;
00BE 3023           00316         movlw   R1
00BF 0084           00317         movwf   FSR
00C0 20C5           00318         call    adjBCD
                    00319 ;
00C1 3022           00320         movlw   R0
00C2 0084           00321         movwf   FSR
00C3 20C5           00322         call    adjBCD
                    00323 ;
00C4 28B3           00324         goto    loop16
                    00325 ;
00C5 3003           00326 adjBCD  movlw   3
00C6 0700           00327         addwf   0,W
00C7 00A7           00328         movwf   temp
00C8 19A7           00329         btfsc   temp,3          ; test if result > 7
00C9 0080           00330         movwf   0
00CA 3030           00331         movlw   30
00CB 0700           00332         addwf   0,W
00CC 00A7           00333         movwf   temp
00CD 1BA7           00334         btfsc   temp,7          ; test if result > 7
00CE 0080           00335         movwf   0               ; save as MSD
00CF 3400           00336         RETLW   0
                    00337 ;
                    00338 ;
                    00339
                    00340         end

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   205
Program Memory Words Free:   819


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     7 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX D

```
MPASM 01.40 Released          MPLXAD.ASM   1-16-1997  16:23:40         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;*******************************************************************
                     00002 ;This program demonstrates how to multiplex four 7 segment LED
                     00003 ;digits and a 4X4 keypad along with 4 A/D inputs using a PIC16C71.
                     00004 ;The four digits will first display the decimal a/d value of ch0.
                     00005 ;When keys from 0 - 3 are hit the corresponding channel's a/d value
                     00006 ;is displayed in decimal.
                     00007 ;The LEDs are updated every 20mS, the keypad is scanned at a rate of 20
                     00008 ;mS. All 4 channels are scanned at 20mS rate, so each channel gets
                     00009 ;scanned every 80mS. A faster rate of scanning is possible as required
                     00010 ;by the users application.
                     00011 ;Timer0 is used in internal interrupt mode to generate the
                     00012 ;5 mS.
                     00013 ;
                     00014 ;                                        Stan D'Souza 5/8/93
                     00015 ;
                     00016 ;Corrected error in display routine.
                     00017 ;                                        Stan D'Souza 2/27/94
                     00018 ;
                     00019 ;       Program:          MPLXAD.ASM
                     00020 ;       Revision Date:
                     00021 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00022 ;
                     00023 ;*******************************************************************
                     00024         LIST P=16C71
                     00025         ERRORLEVEL  -302
                     00026 ;
                     00027         include      <p16c71.inc>
                     00001         LIST
                     00002 ; P16C71.INC Standard Header File, Ver. 1.00 Microchip Technology, Inc.
                     00142         LIST
                     00028 ;
0000000C             00029 TempC   equ       0x0c            ;temp general purpose regs
0000000D             00030 TempD   equ       0x0d
0000000E             00031 TempE   equ       0x0e
00000020             00032 PABuf   equ       0x20
00000021             00033 PBBuf   equ       0x21
0000000F             00034 Count   equ       0x0f            ;count
00000010             00035 MsdTime equ       0x10            ;most significant Timer
00000011             00036 LsdTime equ       0x11            ;Least significant Timer
                     00037 ;
00000012             00038 Flag    equ       0x12            ;general purpose flag reg
                     00039 #define keyhit    Flag,0          ;bit 0 --> key-press on
                     00040 #define DebnceOn  Flag,1          ;bit 1 -> debounce on
                     00041 #define noentry   Flag,2          ;no key entry = 0
                     00042 #define ServKey   Flag,3          ;bit 3 --> service key
                     00043 #define ADOver    Flag,4          ;bit 4 --> a/d conv. over
                     00044 ;
00000013             00045 Debnce  equ       0x13            ;debounce counter
00000014             00046 NewKey  equ       0x14
00000015             00047 DisplayCh equ     0x15            ;channel to be displayed
                     00048 ;
00000016             00049 ADTABLE equ       0x16            ;4 locations are reserved here
                     00050                                   ;from 0x16 to 0x19
                     00051 ;
```

```
  0000002F          00052 WBuffer equ     0x2f
  0000002E          00053 StatBuffer equ  0x2e
  00000001          00054 OptionReg equ   1
  00000002          00055 PCL     equ     2
                    00056 ;
                    00057 ;
                    00058 push    macro
                    00059         movwf   WBuffer         ;save w reg in Buffer
                    00060         swapf   WBuffer, F      ;swap it
                    00061         swapf   STATUS,W        ;get status
                    00062         movwf   StatBuffer      ;save it
                    00063         endm
                    00064 ;
                    00065 pop     macro
                    00066         swapf   StatBuffer,W    ;restore status
                    00067         movwf   STATUS          ;       /
                    00068         swapf   WBuffer,W       ;restore W reg
                    00069         endm
                    00070 ;
0000                00071         org     0
0000 280D           00072         goto    Start           ;skip over interrupt vector
                    00073 ;
0004                00074         org     4
                    00075 ;It is always a good practice to save and restore the w reg,
                    00076 ;and the status reg during a interrupt.
                    00077         push
0004 00AF               M         movwf   WBuffer         ;save w reg in Buffer
0005 0EAF               M         swapf   WBuffer, F      ;swap it
0006 0E03               M         swapf   STATUS,W        ;get status
0007 00AE               M         movwf   StatBuffer      ;save it
0008 2052           00078         call    ServiceInterrupts
                    00079         pop
0009 0E2E               M         swapf   StatBuffer,W    ;restore status
000A 0083               M         movwf   STATUS          ;       /
000B 0E2F               M         swapf   WBuffer,W       ;restore W reg
000C 0009           00080         retfie
                    00081 ;
000D                00082 Start
000D 203B           00083         call    InitPorts
000E 20EE           00084         call    InitAd
000F 2045           00085         call    InitTimers
0010                00086 loop
0010 1992           00087         btfsc   ServKey         ;key service pending
0011 2015           00088         call    ServiceKey      ;yes then service
0012 1A12           00089         btfsc   ADOver          ;a/d pending?
0013 2028           00090         call    ServiceAD       ;yes the service a/d
0014 2810           00091         goto    loop
                    00092 ;
                    00093 ;ServiceKey, does the software service for a keyhit. After a key
                    00094 ;service, the ServKey flag is reset, to denote a completed operation.
0015                00095 ServiceKey
0015 1192           00096         bcf     ServKey         ;reset service flag
0016 0814           00097         movf    NewKey,W        ;get key value
0017 3C03           00098         sublw   3               ;key > 3?
0018 1C03           00099         btfss   STATUS,C        ;no then skip
0019 0008           00100         return                  ;else ignore key
001A 0814           00101         movf    NewKey,W
001B 0095           00102         movwf   DisplayCh       ;load new channel
                    00103 ;
001C                00104 LoadAD
001C 3016           00105         movlw   ADTABLE         ;get top of table
001D 0715           00106         addwf   DisplayCh,W     ;add offset
001E 0084           00107         movwf   FSR             ;init FSR
001F 0800           00108         movf    0,W             ;get a/d value
0020 00A1           00109         movwf   L_byte
0021 01A0           00110         clrf    H_byte
```


© 1997 Microchip Technology Inc.

```
0022 2106          00111         call    B2_BCD
0023 0824          00112         movf    R2,W            ;get LSd
0024 0091          00113         movwf   LsdTime         ;save in LSD
0025 0823          00114         movf    R1,W            ;get Msd
0026 0090          00115         movwf   MsdTime         ;save in Msd
0027 0008          00116         return
                   00117 ;
                   00118 ;This routine essentially loads the ADRES value in the table location
                   00119 ;determined by the channel offset. If channel 0 then ADRES is saved
                   00120 ;in location ADTABLE. If channel 1 then ADRES is saved at ADTABLE + 1.
                   00121 ;and so on.
0028               00122 ServiceAD
0028 0808          00123         movf    ADCON0,W        ;get adcon0
0029 008C          00124         movwf   TempC           ;save in temp
002A 3008          00125         movlw   B'00001000'     ;select next channel
002B 0708          00126         addwf   ADCON0,W        ;       /
002C 1A88          00127         btfsc   ADCON0,5        ;if <= ch3
002D 30C1          00128         movlw   B'11000001'     ;select ch0
002E 0088          00129         movwf   ADCON0
                   00130 ;now load adres in the table
002F 3016          00131         movlw   ADTABLE
0030 0084          00132         movwf   FSR             ;load FSR with top
0031 0C8C          00133         rrf     TempC, F
0032 0C8C          00134         rrf     TempC, F
0033 0C0C          00135         rrf     TempC,W         ;get in w reg
0034 3903          00136         andlw   3               ;mask off all but last 2
0035 0784          00137         addwf   FSR, F          ;add offset to table
0036 0809          00138         movf    ADRES,W         ;get a/d value
0037 0080          00139         movwf   0               ;load indirectly
0038 1212          00140         bcf     ADOver          ;clear flag
0039 201C          00141         call    LoadAD          ;load a/d value in display reg.
003A 0008          00142         return
                   00143
                   00144
                   00145
                   00146 ;
003B               00147 InitPorts
003B 1683          00148         bsf     STATUS,RP0      ;select Bank1
003C 3003          00149         movlw   3               ;make RA0-3 digital I/O
003D 0088          00150         movwf   ADCON1          ;       /
003E 0185          00151         clrf    TRISA           ;make RA0-4 outputs
003F 0186          00152         clrf    TRISB           ;make RB0-7 outputs
0040 1283          00153         bcf     STATUS,RP0      ;select Bank0
0041 0185          00154         clrf    PORTA           ;make all outputs low
0042 0186          00155         clrf    PORTB           ;       /
0043 1585          00156         bsf     PORTA,3         ;enable MSB digit sink
0044 0008          00157         return
                   00158 ;
                   00159 ;
                   00160 ;The clock speed is 4.096Mhz. Dividing internal clk. by a 32 prescaler,
                   00161 ;the TMR0 will be incremented every 31.25uS. If TMR0 is preloaded
                   00162 ;with 96, it will take (256-96)*31.25uS to overflow i.e. 5mS. So the
                   00163 ;end result is that we get a TMR0 interrupt every 5mS.
0045               00164 InitTimers
0045 0190          00165         clrf    MsdTime         ;clr timers
0046 0191          00166         clrf    LsdTime         ;       /
0047 0195          00167         clrf    DisplayCh       ;show channel 0
0048 0192          00168         clrf    Flag            ;clr all flags
0049 1683          00169         bsf     STATUS,RP0      ;select Bank1
004A 3084          00170         movlw   B'10000100'     ;assign ps to TMR0
004B 0081          00171         movwf   OptionReg       ;ps = 32
004C 1283          00172         bcf     STATUS,RP0      ;select Bank0
004D 3020          00173         movlw   B'00100000'     ;enable TMR0 interrupt
004E 008B          00174         movwf   INTCON          ;
004F 3060          00175         movlw   .96             ;preload TMR0
0050 0081          00176         movwf   TMR0            ;start counter
```

```
0051 0009           00177          retfie
                    00178 ;
0052                00179 ServiceInterrupts
0052 190B           00180          btfsc   INTCON,T0IF      ;TMR0 interrupt?
0053 2857           00181          goto    ServiceTMR0      ;yes then service
0054 018B           00182          clrf    INTCON           ;else clr all int
0055 168B           00183          bsf     INTCON,T0IE
0056 0008           00184          return
                    00185 ;
0057                00186 ServiceTMR0
0057 3060           00187          movlw   .96              ;initialize TMR0
0058 0081           00188          movwf   TMR0
0059 110B           00189          bcf     INTCON,T0IF      ;clr int flag
005A 1805           00190          btfsc   PORTA,0          ;scan keys every 20 mS
005B 2060           00191          call    ScanKeys         ;when digit 1 is on
005C 1985           00192          btfsc   PORTA,3          ;scan a/d every 20mS
005D 20F1           00193          call    SampleAd         ;when digit 4 is on
005E 20BF           00194          call    UpdateDisplay    ;update display
005F 0008           00195          return
                    00196 ;
                    00197 ;
                    00198 ;ScanKeys, scans the 4x4 keypad matrix and returns a key value in
                    00199 ;NewKey (0 - F) if a key is pressed, if not it clears the keyhit flag.
                    00200 ;Debounce for a given keyhit is also taken care of.
                    00201 ;The rate of key scan is 20mS with a 4.096Mhz clock.
0060                00202 ScanKeys
0060 1C92           00203          btfss   DebnceOn         ;debounce on?
0061 2866           00204          goto    Scan1            ;no then scan keypad
0062 0B93           00205          decfsz  Debnce, F        ;else dec debounce time
0063 0008           00206          return                   ;not over then return
0064 1092           00207          bcf     DebnceOn         ;over, clr debounce flag
0065 0008           00208          return                   ;and return
0066                00209 Scan1
0066 20A8           00210          call    SavePorts        ;save port values
0067 30EF           00211          movlw   B'11101111'      ;init TempD
0068 008D           00212          movwf   TempD
0069                00213 ScanNext
0069 0806           00214          movf    PORTB,W          ;read to init port
006A 100B           00215          bcf     INTCON,RBIF      ;clr flag
006B 0C8D           00216          rrf     TempD, F         ;get correct column
006C 1C03           00217          btfss   STATUS,C         ;if carry set?
006D 2880           00218          goto    NoKey            ;no then end
006E 080D           00219          movf    TempD,W          ;else output
006F 0086           00220          movwf   PORTB            ;low column scan line
0070 0000           00221          nop
0071 1C0B           00222          btfss   INTCON,RBIF      ;flag set?
0072 2869           00223          goto    ScanNext         ;no then next
0073 1812           00224          btfsc   keyhit           ;last key released?
0074 287E           00225          goto    SKreturn         ;no then exit
0075 1412           00226          bsf     keyhit           ;set new key hit
0076 0E06           00227          swapf   PORTB,W          ;read port
0077 008E           00228          movwf   TempE            ;save in TempE
0078 2082           00229          call    GetKeyValue      ;get key value 0 - F
0079 0094           00230          movwf   NewKey           ;save as New key
007A 1592           00231          bsf     ServKey          ;set service flag
007B 1492           00232          bsf     DebnceOn         ;set flag
007C 3004           00233          movlw   4
007D 0093           00234          movwf   Debnce           ;load debounce time
007E                00235 SKreturn
007E 20B5           00236          call    RestorePorts     ;restore ports
007F 0008           00237          return
                    00238 ;
0080                00239 NoKey
0080 1012           00240          bcf     keyhit           ;clr flag
0081 287E           00241          goto    SKreturn
                    00242 ;
```


© 1997 Microchip Technology Inc.

```
                    00243 ;GetKeyValue gets the key as per the following layout
                    00244 ;
                    00245 ;                    Col1    Col2    Col3    Col4
                    00246 ;                    (RB3)   (RB2)   (RB1)   (RB0)
                    00247 ;
                    00248 ;Row1(RB4)             0       1       2       3
                    00249 ;
                    00250 ;Row2(RB5)             4       5       6       7
                    00251 ;
                    00252 ;Row3(RB6)             8       9       A       B
                    00253 ;
                    00254 ;Row4(RB7)             C       D       E       F
                    00255 ;
0082                00256 GetKeyValue
0082 018C           00257         clrf    TempC
0083 1D8D           00258         btfss   TempD,3         ;first column
0084 288C           00259         goto    RowValEnd
0085 0A8C           00260         incf    TempC, F
0086 1D0D           00261         btfss   TempD,2         ;second col.
0087 288C           00262         goto    RowValEnd
0088 0A8C           00263         incf    TempC, F
0089 1C8D           00264         btfss   TempD,1         ;3rd col.
008A 288C           00265         goto    RowValEnd
008B 0A8C           00266         incf    TempC, F        ;last col.
008C                00267 RowValEnd
008C 1C0E           00268         btfss   TempE,0         ;top row?
008D 2896           00269         goto    GetValCom       ;yes then get 0,1,2&3
008E 1C8E           00270         btfss   TempE,1         ;2nd row?
008F 2895           00271         goto    Get4567         ;yes the get 4,5,6&7
0090 1D0E           00272         btfss   TempE,2         ;3rd row?
0091 2893           00273         goto    Get89ab         ;yes then get 8,9,a&b
0092                00274 Getcdef
0092 150C           00275         bsf     TempC,2         ;set msb bits
0093                00276 Get89ab
0093 158C           00277         bsf     TempC,3         ;       /
0094 2896           00278         goto    GetValCom       ;do common part
0095                00279 Get4567
0095 150C           00280         bsf     TempC,2
0096                00281 GetValCom
0096 080C           00282         movf    TempC,W
0097 0782           00283         addwf   PCL, F
0098 3400           00284         retlw   0
0099 3401           00285         retlw   1
009A 3402           00286         retlw   2
009B 3403           00287         retlw   3
009C 3404           00288         retlw   4
009D 3405           00289         retlw   5
009E 3406           00290         retlw   6
009F 3407           00291         retlw   7
00A0 3408           00292         retlw   8
00A1 3409           00293         retlw   9
00A2 340A           00294         retlw   0a
00A3 340B           00295         retlw   0b
00A4 340C           00296         retlw   0c
00A5 340D           00297         retlw   0d
00A6 340E           00298         retlw   0e
00A7 340F           00299         retlw   0f
                    00300 ;
                    00301 ;SavePorts, saves the porta and portb condition during a key scan
                    00302 ;operation.
00A8                00303 SavePorts
00A8 0805           00304         movf    PORTA,W         ;Get sink value
00A9 00A0           00305         movwf   PABuf           ;save in buffer
00AA 0185           00306         clrf    PORTA           ;disable all sinks
00AB 0806           00307         movf    PORTB,W         ;get port b
00AC 00A1           00308         movwf   PBBuf           ;save in buffer
```


© 1997 Microchip Technology Inc.

```
00AD 30FF           00309         movlw   0xff           ;make all high
00AE 0086           00310         movwf   PORTB          ;on port b
00AF 1683           00311         bsf     STATUS,RP0     ;select Bank1
00B0 1381           00312         bcf     OptionReg,7    ;enable pull ups
00B1 30F0           00313         movlw   B'11110000'    ;port b hi nibble inputs
00B2 0086           00314         movwf   TRISB          ;lo nibble outputs
00B3 1283           00315         bcf     STATUS,RP0     ;Bank0
00B4 0008           00316         return
                    00317 ;
                    00318 ;RestorePorts, restores the condition of porta and portb after a
                    00319 ;key scan operation.
00B5                00320 RestorePorts
00B5 0821           00321         movf    PBBuf,W        ;get port b
00B6 0086           00322         movwf   PORTB
00B7 0820           00323         movf    PABuf,W        ;get port a value
00B8 0085           00324         movwf   PORTA
00B9 1683           00325         bsf     STATUS,RP0     ;select Bank1
00BA 1781           00326         bsf     OptionReg,7    ;disable pull ups
00BB 0185           00327         clrf    TRISA          ;make port a outputs
00BC 0186           00328         clrf    TRISB          ;as well as PORTB
00BD 1283           00329         bcf     STATUS,RP0     ;Bank0
00BE 0008           00330         return
                    00331 ;
                    00332 ;
00BF                00333 UpdateDisplay
00BF 0805           00334         movf    PORTA,W        ;present sink value in w
00C0 0185           00335         clrf    PORTA          ;disable all digits sinks
00C1 390F           00336         andlw   0x0f
00C2 008C           00337         movwf   TempC          ;save sink value in tempC
00C3 160C           00338         bsf     TempC,4        ;preset for lsd sink
00C4 0C8C           00339         rrf     TempC, F       ;determine next sink value
00C5 1C03           00340         btfss   STATUS,C       ;c=1?
00C6 118C           00341         bcf     TempC,3        ;no then reset LSD sink
00C7 180C           00342         btfsc   TempC,0        ;else see if Msd
00C8 28D6           00343         goto    UpdateMsd      ;yes then do Msd
00C9 188C           00344         btfsc   TempC,1        ;see if 3rdLsd
00CA 28D3           00345         goto    Update3rdLsd   ;yes then do 3rd Lsd
00CB 190C           00346         btfsc   TempC,2        ;see if 2nd Lsd
00CC 28D0           00347         goto    Update2ndLsd   ;yes then do 2nd lsd
00CD                00348 UpdateLsd
00CD 0811           00349         movf    LsdTime,W      ;get Lsd in w
00CE 390F           00350         andlw   0x0f           ;       /
00CF 28D8           00351         goto    DisplayOut
00D0                00352 Update2ndLsd
00D0 0E11           00353         swapf   LsdTime,W      ;get 2nd Lsd in w
00D1 390F           00354         andlw   0x0f           ;mask rest
00D2 28D8           00355         goto    DisplayOut     ;enable display
00D3                00356 Update3rdLsd
00D3 0810           00357         movf    MsdTime,W      ;get 3rd Lsd in w
00D4 390F           00358         andlw   0x0f           ;mask low nibble
00D5 28D8           00359         goto    DisplayOut     ;enable display
00D6                00360 UpdateMsd
00D6 0E10           00361         swapf   MsdTime,W      ;get Msd in w
00D7 390F           00362         andlw   0x0f           ;mask rest
00D8                00363 DisplayOut
00D8 20DD           00364         call    LedTable       ;get digit output
00D9 0086           00365         movwf   PORTB          ;drive leds
00DA 080C           00366         movf    TempC,W        ;get sink value in w
00DB 0085           00367         movwf   PORTA
00DC 0008           00368         return
                    00369 ;
                    00370 ;
00DD                00371 LedTable
00DD 0782           00372         addwf   PCL, F         ;add to PC low
00DE 343F           00373         retlw   B'00111111'    ;led drive for 0
00DF 3406           00374         retlw   B'00000110'    ;led drive for 1
```


© 1997 Microchip Technology Inc.

```
00E0 345B          00375         retlw   B'01011011'    ;led drive for 2
00E1 344F          00376         retlw   B'01001111'    ;led drive for 3
00E2 3466          00377         retlw   B'01100110'    ;led drive for 4
00E3 346D          00378         retlw   B'01101101'    ;led drive for 5
00E4 347D          00379         retlw   B'01111101'    ;led drive for 6
00E5 3407          00380         retlw   B'00000111'    ;led drive for 7
00E6 347F          00381         retlw   B'01111111'    ;led drive for 8
00E7 3467          00382         retlw   B'01100111'    ;led drive for 9
00E8 3477          00383         retlw   B'01110111'    ;led drive for A
00E9 347C          00384         retlw   B'01111100'    ;led drive for b
00EA 3439          00385         retlw   B'00111001'    ;led drive for C
00EB 345E          00386         retlw   B'01011110'    ;led drive for d
00EC 3479          00387         retlw   B'01111001'    ;led drive for E
00ED 3471          00388         retlw   B'01110001'    ;led drive for F
                   00389
                   00390 ;
                   00391 ;
00EE               00392 InitAd
00EE 30C0          00393         movlw   B'11000000'    ;internal rc for tad
00EF 0088          00394         movwf   ADCON0         ;       /
                   00395         ;note that adcon1 is set in InitPorts
00F0 0008          00396         return
                   00397 ;
00F1               00398 SampleAd
00F1 20A8          00399         call    SavePorts
00F2 20F8          00400         call    DoAd             ;do a ad conversion
00F3               00401 AdDone
00F3 1908          00402         btfsc   ADCON0,GO        ;ad done?
00F4 28F3          00403         goto    AdDone           ;no then loop
00F5 1612          00404         bsf     ADOver           ;set a/d over flag
00F6 20B5          00405         call    RestorePorts     ;restore ports
00F7 0008          00406         return
                   00407 ;
                   00408 ;
00F8               00409 DoAd
00F8 0186          00410         clrf    PORTB            ;turn off leds
00F9 1683          00411         bsf     STATUS,RP0       ;select Bank1
00FA 300F          00412         movlw   0x0f             ;make port a hi-Z
00FB 0085          00413         movwf   TRISA            ;       /
00FC 1283          00414         bcf     STATUS,RP0       ;select Bank0
00FD 1408          00415         bsf     ADCON0,ADON      ;start a/d
00FE 307D          00416         movlw   .125
00FF 2102          00417         call    Wait
0100 1508          00418         bsf     ADCON0,GO        ;start conversion
0101 0008          00419         return
                   00420 ;
                   00421 ;
0102               00422 Wait
0102 008C          00423         movwf   TempC            ;store in temp
0103               00424 Next
0103 0B8C          00425         decfsz  TempC, F
0104 2903          00426         goto    Next
0105 0008          00427         return
                   00428
                   00429 ;
                   00430 ;
  00000026         00431 count   equ     26
  00000027         00432 temp    equ     27
                   00433 ;
  00000020         00434 H_byte  equ     20
  00000021         00435 L_byte  equ     21
  00000022         00436 R0      equ     22               ; RAM Assignments
  00000023         00437 R1      equ     23
  00000024         00438 R2      equ     24
                   00439 ;
                   00440 ;
```

```
0106 1003           00441 B2_BCD  bcf     STATUS,0        ; clear the carry bit
0107 3010           00442         movlw   .16
0108 00A6           00443         movwf   count
0109 01A2           00444         clrf    R0
010A 01A3           00445         clrf    R1
010B 01A4           00446         clrf    R2
010C 0DA1           00447 loop16  rlf     L_byte, F
010D 0DA0           00448         rlf     H_byte, F
010E 0DA4           00449         rlf     R2, F
010F 0DA3           00450         rlf     R1, F
0110 0DA2           00451         rlf     R0, F
                    00452 ;
0111 0BA6           00453         decfsz  count, F
0112 2914           00454         goto    adjDEC
0113 3400           00455         RETLW   0
                    00456 ;
0114 3024           00457 adjDEC  movlw   R2
0115 0084           00458         movwf   FSR
0116 211E           00459         call    adjBCD
                    00460 ;
0117 3023           00461         movlw   R1
0118 0084           00462         movwf   FSR
0119 211E           00463         call    adjBCD
                    00464 ;
011A 3022           00465         movlw   R0
011B 0084           00466         movwf   FSR
011C 211E           00467         call    adjBCD
                    00468 ;
011D 290C           00469         goto    loop16
                    00470 ;
011E 3003           00471 adjBCD  movlw   3
011F 0700           00472         addwf   0,W
0120 00A7           00473         movwf   temp
0121 19A7           00474         btfsc   temp,3          ; test if result > 7
0122 0080           00475         movwf   0
0123 3030           00476         movlw   30
0124 0700           00477         addwf   0,W
0125 00A7           00478         movwf   temp
0126 1BA7           00479         btfsc   temp,7          ; test if result > 7
0127 0080           00480         movwf   0               ; save as MSD
0128 3400           00481         RETLW   0
                    00482 ;
                    00483 ;
                    00484 ;
                    00485 ;
                    00486
                    00487         end


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXX------- ----------------

All other memory blocks unused.

Program Memory Words Used:   294
Program Memory Words Free:   730


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     7 suppressed
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN566

Using the PORTB Interrupt on Change as an External Interrupt

Author: *Mark Palmer*
Microchip Technology Inc.

INTRODUCTION

The PICmicro™ families of RISC microcontrollers are designed to provide advanced performance and a cost-effective solution for a variety of applications. To address these applications, there is the PIC16CXXX microcontroller family of products. This family has numerous peripheral and special features to better address user applications.

The feature this application note will focus on is the Interrupt on Change of the PORTB pins. This "interrupt on change" is triggered when any of the RB7:RB4 pins, configured as an input, changes level. When this interrupt is used in conjunction with the software programmable weak internal pull-ups, a direct interface to a keypad is possible. This is shown in application note AN552, Implementing Wake-up on Key Stroke. Another way to use the "interrupt on change" feature would be as additional external interrupt sources. This allows PIC16CXXX devices to support multiple external interrupts, in addition to the built-in external interrupt on the INT pin.

This application note will discuss some of the issues in using PORTB as additional external interrupt pins, and will show some examples. These examples can be easily modified to suit your particular needs.

USING A PORTB INPUT FOR AN EXTERNAL INTERRUPT

The interrupt source(s) cannot simply be directly connected to the PORTB pins, and expect an interrupt to occur the same as on the interrupt (INT) pin. To develop the microcontrollers hardware/software to act as an interrupt by an external signal, we must know the characteristics of the external signal. After we know this, we can determine the best way to structure the program to handle this signal. The characteristics that we need to consider when developing the interrupt include:

1. The rising edge and falling edges.
2. The pulse width of the interrupt trigger (high time / low time).

It is easy to understand the need of knowing about which edge triggers the interrupt service routine for the external interrupt. This allows one to ensure that the interrupt service routine is only entered for the desired edge, with all other edges ignored. Not so clear is the pulse width of the interrupt's trigger. This characteristic helps determine the amount of additional overhead that the software routine may need.

© 1997 Microchip Technology Inc.

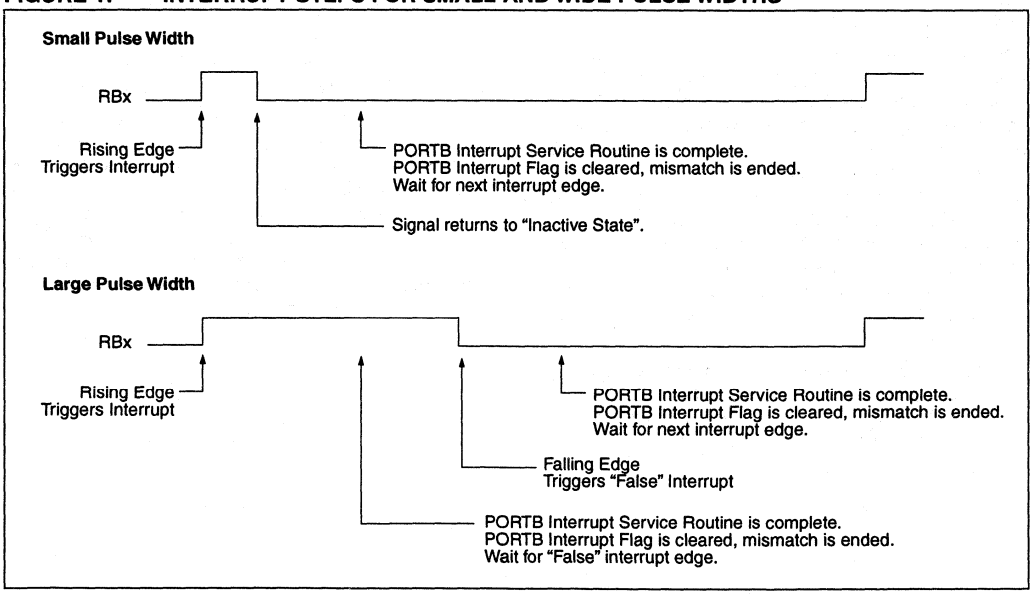
Figure 1 shows the two cases for the interrupt signal verses the time to complete the interrupt service routine. The first waveform is when the signal makes the low-to-high-to-low transitions before the interrupt service routine has completed (interrupt flag cleared). When the interrupt flag has been cleared, the interrupt signal has already returned to the inactive level. The next transition of the signal is due to another interrupt request. An interrupt signal with this characteristic will be called a small pulse width signal.

The second waveform is when the signal only makes the low-to-high transitions before the interrupt service routine has completed (interrupt flag cleared). The next transition (high-to-low) will return the interrupt signal to the inactive level. This will generate a "false" interrupt, that will need to be cleared. Then the following transition (low-to-high) will be a "true" interrupt. An interrupt signal with this characteristic will be called a wide pulse width signal.

An interrupt pulse with a small pulse width requires less overhead than a wide pulse width. A small pulse width signal must be less than the minimum execution time of the interrupt service routine, while a wide pulse width must be greater then the maximum time through the interrupt service routine.

Example 1 shows a single interrupt source on PORTB (RB7), which executes the interrupt service routine on a rising edge. The interrupt source has a small pulse width. In this case, since the interrupt pulse width is small, the pulse has gone high and then low again before PORTB is read to end the mismatch condition. So when PORTB is read it will read a low signal and will again be waiting for the rising edge transition.

FIGURE 1: INTERRUPT STEPS FOR SMALL AND WIDE PULSE WIDTHS



EXAMPLE 1: SINGLE INTERRUPT WITH A SMALL PULSE WIDTH

```
PER_INT         BTFSS   INTCON, RBIF        ; PortB interrupt?
                GOTO    OTHER_INT           ; Other interrupt
                        :                   ; Do task for INT on RB7
                        :                   ;
CLR_RBINTF      MOVF    PORTB, 1            ; Read PortB (to itself) to end
                                            ; mismatch condition
                BCF     INTCON, RBIF        ; Clear the RB interrupt flag.
                RETFIE                      ; Return from interrupt
OTHER_INT               :                   ; Do what you need to here
                        :
                RETFIE                      ; Return from interrupt
```


© 1997 Microchip Technology Inc.

Example 2 shows a single interrupt source on PORTB (RB7), which executes the interrupt service routine on a rising edge. The interrupt source has a wide pulse width. In this case since the interrupt pulse width is large, the pulse is still high before PORTB is read to end the mismatch condition. So when PORTB is read it will read a high signal and will generate an interrupt on the next falling edge transition (which should be ignored).

EXAMPLE 2: SINGLE INTERRUPT WITH A WIDE PULSE WIDTH

```
PER_INT       BTFSS  INTCON, RBIF        ; PortB interrupt?
              GOTO   OTHER_INT           ; Other interrupt
              BTFSC  PORTB, RB7          ; Check for rising edge
              GOTO   CLR_RBINTF          ; Falling edge, clear PortB int
              :                          ;   flag
                     :                   ; Do task for INT on RB7
                     :
CLR_RBINTF    MOVF   PORTB, 1            ; Read PortB (to itself) to end
                                         ;    mismatch condition
              BCF    INTCON, RBIF        ; Clear the RB interrupt flag.
              RETFIE                     ; Return from interrupt
OTHER_INT              :                 ; Do what you need to here
                       :
              RETFIE                     ; Return from interrupt
```

Example 3 shows an interrupt on change with the interrupt source on PORTB (RB7). This executes the interrupt service routine on a both edges. The interrupt source must have a minimum pulse width to ensure that both edges can be "seen". The minimum pulse width is the maximum time from the interrupt edge to the reading of PORTB and clearing the interrupt flag.

EXAMPLE 3: INTERRUPT ON CHANGE

```
PER_INT       BTFSS  INTCON, RBIF        ; PortB interrupt?
              GOTO   OTHER_INT           ; Other interrupt
CLR_RBINTF    MOVF   PORTB, 1            ; Read PortB (to itself) to end
                                         ;    mismatch condition
              BCF    INTCON, RBIF        ; Clear the RB interrupt flag.
                     :                   ; Do task for INT on RB7
                     :                   ;
              RETFIE                     ; Return from interrupt
OTHER_INT                                ; Do what you need to here
                     :
              RETFIE                     ; Return from interrupt
```


© 1997 Microchip Technology Inc.

USING PORTB INPUTS FOR MULTIPLE INTERRUPTS

The previous examples have been for a single external interrupt on PORTB. This can be extended to support up to four external interrupts. To do this requires additional software overhead, to determine which of the PORTB pins (RB7:RB4) caused the interrupt. Care should be taken in the software to ensure that no interrupts are lost.

In this example, the interrupt sources on RB7, RB5, and RB4 have a small pulse width, while the interrupt source on pin RB6 is wide and should cause a trigger on the rising edge.

SUMMARY

The PORTB interrupt on change feature is both a very convenient method for direct interfacing to an external keypad, with no additional components, but is also versatile in its uses the ability to add up to four additional external interrupts. Of course hybrid solutions are also possible. That is, for example, using PORTB<6:1> as a 3x3 keypad, with PORTB<7> as an external interrupt and PORTB<0> as a general purpose I/O. The flexibility of this feature allows the user to implement a best fit design for the application.

EXAMPLE 4: MULTIPLE INTERRUPTS WITH DIFFERENT PULSE WIDTHS

```
PER_INT         BTFSS   INTCON, RBIF        ; PortB interrupt?
                GOTO    OTHER_INT           ; Other interrupt
;
; PortB change interrupt has occurred. Must determine which pin caused
; interrupt and do appropriate action. That is service the interrupt,
; or clear flags due to other edge.
;
                MOVF    PORTB, 0            ; Move PortB value to the W register
                                            ;   This ends mismatch conditions
                MOVWF   TEMP                ; Need to save the PortB reading.
                XORWF   LASTPB, 1           ; XOR last PortB value with the new
                                            ;   PortB value.
CK_RB7          BTFSC   LASTPB, RB7         ; Did pin RB7 change
                CALL    RB7_CHG             ; RB7 changed and caused the interrupt
CK_RB6          BTFSC   LASTPB, RB6         ; Did pin RB6 change
                CALL    RB6_CHG             ; RB6 changed and caused the interrupt
CK_RB5          BTFSC   LASTPB, RB5         ; Did pin RB5 change
                CALL    RB5_CHG             ; RB5 changed and caused the interrupt
CK_RB4          BTFSC   LASTPB, RB4         ; Did pin RB4 change
                GOTO    RB4_CHG             ; RB4 changed and caused the interrupt
;
RB7_CHG             :                       ; Do task for INT on RB7
                    :                       ;
                RETURN
RB6_CHG         BTFSC   PORTB, RB6          ; Check for rising edge
                RETURN                      ; Falling edge, Ignore
                    :                       ; Do task for INT on RB6
                    :
                RETURN
RB5_CHG             :                       ; Do task for INT on RB5
                    :                       ;
                RETURN
RB4_CHG             :                       ; Do task for INT on RB4
                    :                       ;
CLR_RBINTF      MOVF    TEMP, 0             ; Move the PortB read value to the
                MOVWF   LASTPB              ;   register LASTPB
                BCF     INTCON, RBIF        ; Clear the RB interrupt flag.
                RETFIE                      ; Return from interrupt
;
OTHER_INT           :                       ; Do what you need to here
                    :
                RETFIE                      ; Return from interrupt
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN578

Use of the SSP Module in the I²C™ Multi-Master Environment

Author: *Scott Fink*
Microchip Technology Inc.

INTRODUCTION

The Inter-IC (I^2C) bus is a two-wire serial interface developed by Philips/Signetics. The specification supports data transmission up to 400 Kbps.

The I^2C interface employs a comprehensive protocol to ensure reliable transmission and reception of data. When the bus is active, one device is the Master (generates the clock and the handshaking signals), while all the other devices are Slaves. The current bus Master can both read-from and write-to any of the Slave units by addressing them individually. On a Multi-Master bus the Masters follow an arbitration scheme to ensure that the bus is not corrupted.

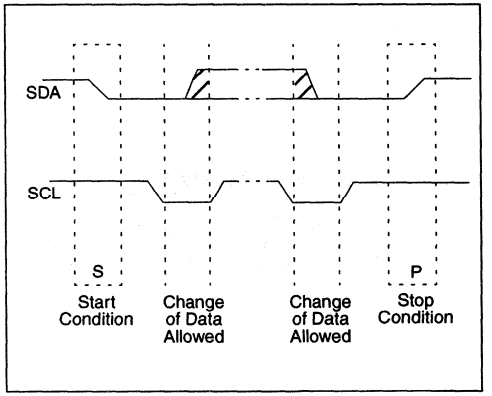
Each device attached to the I^2C bus is assigned a unique address. When a Master wishes to initiate a data transfer, it first transmits the address of the device that it wishes to "talk" to. All devices "listen" to see if this is their address. Within this address, a bit specifies whether the Master wishes to read-from or write-to the Slave device.

The output stages of each device on the bus, attached to the clock (SCL) and data (SDA) lines, must have an open-drain or open-collector in order to perform the wired-AND function of the bus. External pull-up resistors are used to ensure a high level when no device is pulling the line down. The only limitation on the number of devices that may be attached to the bus is the maximum bus loading specification. For complete bus specifications, refer to Philips/Signetics document "The I^2C-bus and How to Use It" (www.semiconductors.philips.com).

INITIATING AND TERMINATING DATA TRANSFER

During times of no data transfer (idle time), both the SCL and SDA lines are pulled high. A Master device which wishes to take control of the bus must first generate a START condition. A START is defined as a high to low transition of SDA when SCL is high. When the Master has completed all data transmissions and wishes to relinquish the bus, it generates a STOP condition. A STOP is defined as a low to high transition of SDA while SCL is high. Because the START and STOP conditions are defined as transitions of the SDA when the SCL line is high, the SDA line can only change when SCL is low during the actual data transmission. Figure 1 shows the relationship between SCL and SDA for the various conditions.

FIGURE 1: START AND STOP CONDITIONS



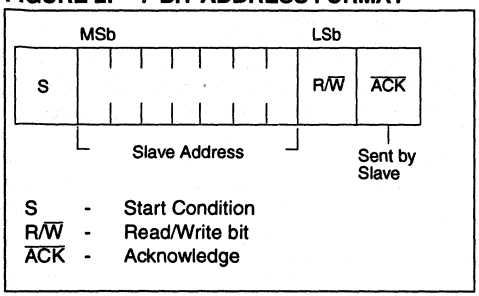
3

© 1997 Microchip Technology Inc.

ADDRESSING I²C DEVICES

There are two address formats. The simplest of these is the 7-bit address format with a R/$\overline{W}$ bit (Figure 2). The more complex is the 10-bit address with a R/$\overline{W}$ bit (Figure 3). For 10-bit addressing, two bytes must be transmitted with the first five bits specifying this to be a 10-bit address. Only 7-bit addressing is used in this application note.

FIGURE 2: 7-BIT ADDRESS FORMAT



TRANSFER ACKNOWLEDGE

Slave as Receiver

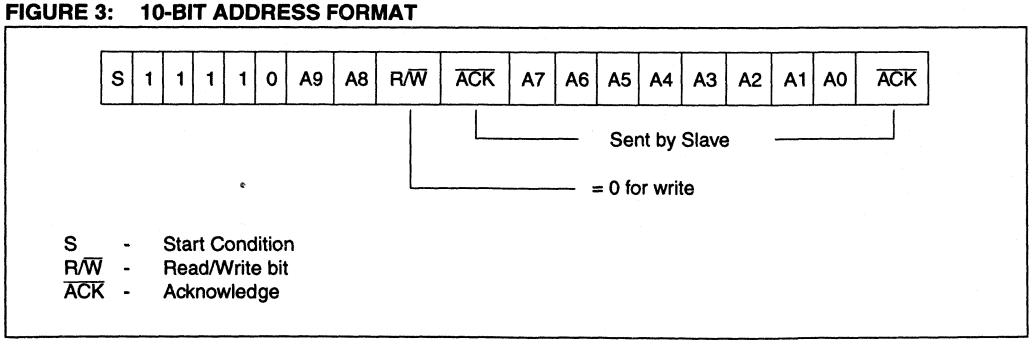
All data is transmitted as bytes, with no limit to the number of bytes transmitted per data transfer. After each byte, the slave-receiver generates an acknowledge bit ($\overline{ACK}$) by pulling the SDA line low. When a slave-receiver doesn't acknowledge the slave address or received data, the master aborts the transfer. Whether the $\overline{ACK}$ bit is generated or not, the SDA line must be released by the slave so that the master can generate the STOP condition.

Master as Receiver

If the master is receiving the data, it generates an acknowledge signal for each received byte of data except for the last byte. To signal the end of data to the slave-transmitter, the master does not generate an acknowledge. The slave then releases the SDA line so the master can generate the STOP condition. The master can also generate the STOP condition during the acknowledge pulse for valid termination of data transfer.

If the slave needs to delay the transmission of the next byte, holding the SCL line low will force the master into a wait state. Data transfer continues when the slave releases the SCL line. This allows the slave to move the received data or fetch the data it needs to transfer before allowing the clock to start. This wait state technique can also be implemented at the bit level.

FIGURE 3: 10-BIT ADDRESS FORMAT



© 1997 Microchip Technology Inc.

MULTI-MASTER

The I^2C protocol allows a system to have more than one master. When two or more masters try to transfer data at the same time, arbitration and synchronization occur.

Arbitration

Arbitration takes place on the SDA line while the SCL line is high. The master which transmits a high when the other master transmits a low loses arbitration (Figure 4) and turns off its data output stage. A master which lost arbitration can generate clock pulses until the end of the data byte where it lost arbitration. When the master devices are addressing the same device, arbitration continues into the data.

Clock Synchronization

Clock synchronization occurs after the devices have started arbitration. This is performed using a wired-AND connection to the SCL line. A high to low transition on the SCL line causes the concerned devices to start counting off their low period. Once a device clock has gone low, it will hold the SCL line low until its SCL high state is reached. The low to high transition of this clock may not change the state of the SCL line, if another device clock is still within its low period. The SCL line is held low by the device with the longest low period. Devices with shorter low periods enter a high wait-state, until the SCL line comes high. When the SCL line comes high, all devices start counting off their high periods. The first device to complete its high period will pull the SCL line low. The SCL line high time is determined by the device with the shortest high period (Figure 5).

3

FIGURE 4: MULTI-MASTER ARBITRATION

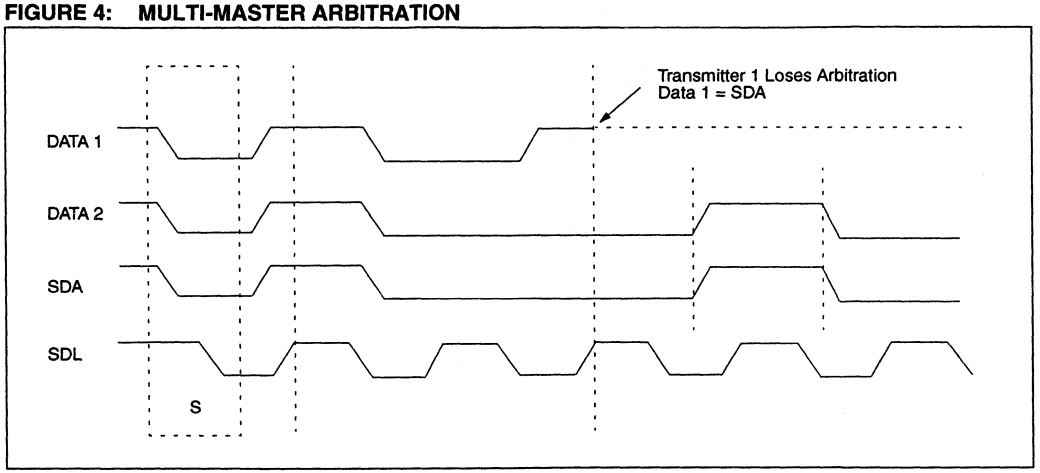
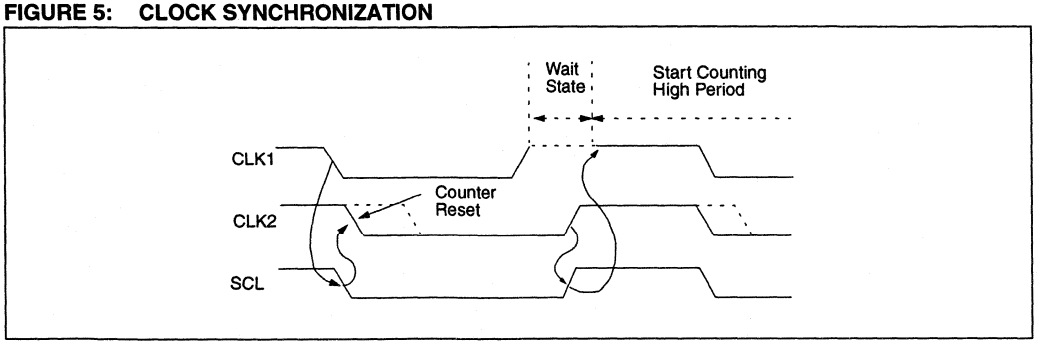


FIGURE 5: CLOCK SYNCHRONIZATION



© 1997 Microchip Technology Inc.

IMPLEMENTATION IN THE PIC16CXXX

This Application Note uses the PIC16CXXX in a Multi-Master I^2C environment. The PIC16CXXX acts as both a Master and a Slave on the bus.

Hardware

The demonstration hardware consists of a keypad multiplexed with eight LEDs on PORTB and connections to the I^2C bus through the RC3/SCL and RC4/SDA pins (Figure 6).

Software

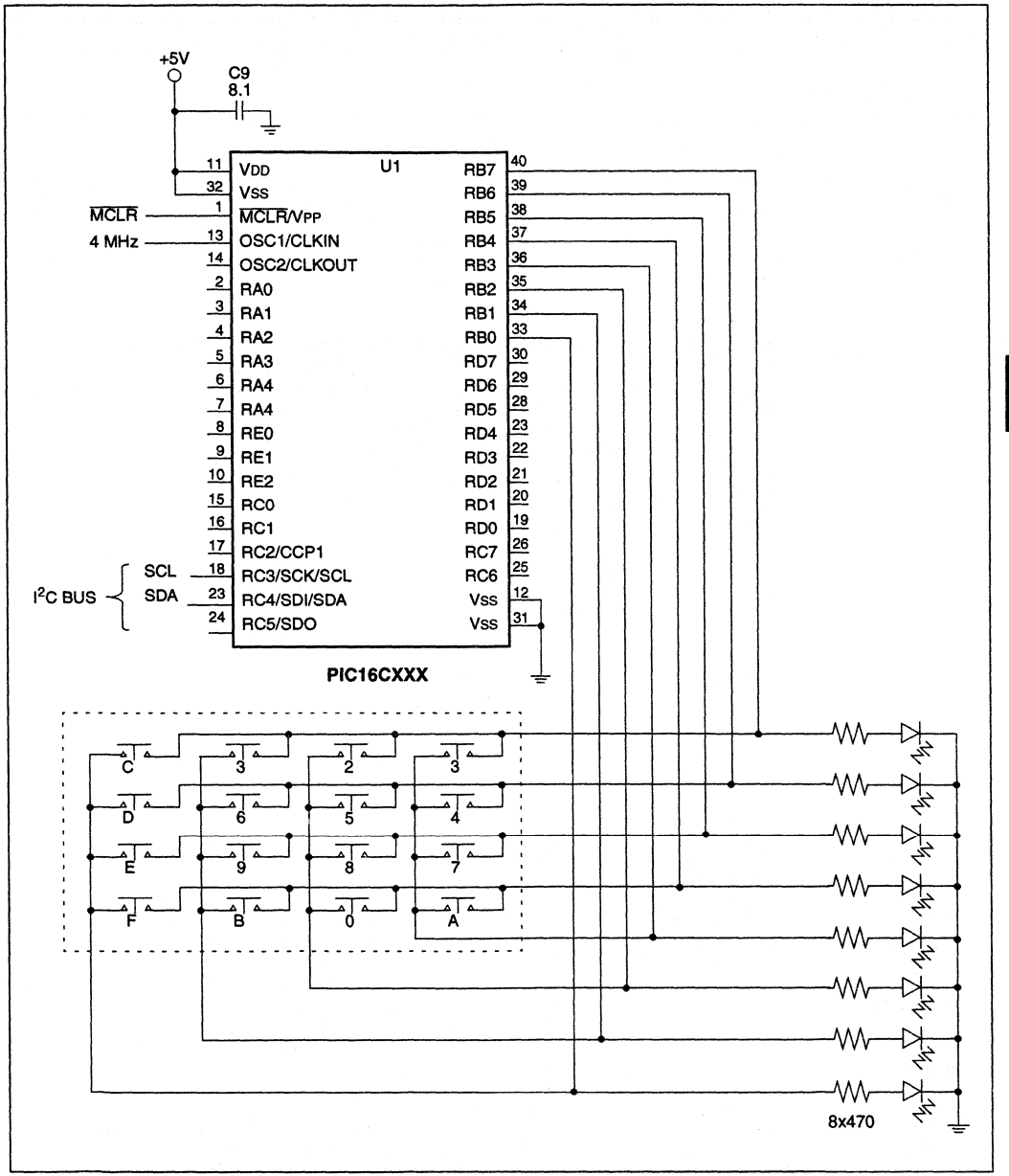
The software transmits in Master mode and receives data in Slave mode.

The **transmit** routine scans the keypad, debounces the keypresses, and transmits their encoded value in Master mode over the I^2C bus to slave address A6. Before transmitting, the status of the Synchronous Serial Port (SSP) is checked. No data is sent until the SSP status register indicates that a Stop bit was received. If the transmission causes any errors, the error code will be displayed on the LEDs, and transmission will again be attempted.

Data **received** in Slave mode at address A2 from the bus is displayed on the LEDs. Slave reception is interrupt driven. When a complete word is received with the proper address, the processor is interrupted, and the program verifies that an SSP interrupt was received. Once it has been verified that the interrupt was caused by the SSP module, and that the buffer is full, the data word is read and output to the LEDs.

© 1997 Microchip Technology Inc.

FIGURE 6: HARDWARE



Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40 Released          IICMULT.ASM   1-16-1997  17:01:20         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         TITLE " Demonstration of I2C MultiMaster mode"
                     00002 ;
                     00003                 LIST P=16C64
                     00004                 ERRORLEVEL  -302
                     00005 ;
                     00006 ;*****************************************************************
                     00007 ;**  Two wire/I2C Bus MultiMaster sample routines for Microchip's
                     00008 ;**  PIC16C64 8-bit CMOS single chip microcomputer
                     00009 ;**  Revised Version  (3/06/94).
                     00010 ;       Program:          IICMULT.ASM
                     00011 ;       Revision Date:
                     00012 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00013 ;
                     00014 ;**
                     00015 ;**     Part used = PIC16C64
                     00016 ;**Note: 1)All timings are based on a reference crystal frequency of
                     00017 ;**    4MHz which is equivalent to an instruction cycle time of 1 usec.
                     00018 ;**      2) Address and literal values are read in decimal unless
                     00019 ;**        otherwise specified.
                     00020 ;*****************************************************************
                     00021 ;
                     00022 ;-----------------------------------------------------------------
                     00023 ;       File Register Assignment
                     00024 ;-----------------------------------------------------------------
                     00025 ;
                     00026         include <p16C64.inc>
                     00001         LIST
                     00002 ;P16C64.INC Standard Header File, Version 1.01 Microchip Technology
                     00238         LIST
                     00027 ;
  00000020           00028 FLAG            EQU     20h     ; Common flag bits register
  00000021           00029 EEPROM          EQU     21h     ; Bit buffer
  00000022           00030 ERCODE          EQU     22h     ; Error code (to indicate bus status)
  00000023           00031 ADDR            EQU     23h     ; Address register
  00000024           00032 DATAI           EQU     24h     ; Stored data input register
  00000025           00033 DATAO           EQU     25h     ; Stored data output register
  00000026           00034 SLAVE           EQU     26h     ; Device address (1010xxx0)
  00000027           00035 TXBUF           EQU     27h     ; TX buffer
  00000028           00036 RXBUF           EQU     28h     ; RX buffer
  00000029           00037 COUNT           EQU     29h     ; Bit counter
  0000002A           00038 TEMP            EQU     2Ah     ; Temporary storage
  0000002B           00039 ROW             EQU     2Bh     ; Keypad row
  0000002C           00040 NEW_KEY         EQU     2Ch     ; Storage for latest key
  0000002D           00041 OLD_KEY         EQU     2Dh     ; Storage for last key pressed
  0000002E           00042 DISPVAL         EQU     2Eh     ; Value displayed on LEDs
  0000002F           00043 TEMP1           EQU     2Fh     ; Scratchpad register
  00000030           00044 TEMP_W          EQU     30h     ; Storage for W register
  00000031           00045 TEMP_STAT       EQU     31h     ; Storage for STATUS register
                     00046 ;
                     00047 ;-----------------------------------------------------------------
                     00048 ;                     Bit Assignments
                     00049 ;-----------------------------------------------------------------
                     00050
                     00051 ; FLAG Bits
```


© 1997 Microchip Technology Inc.

```
                    00052
  00000000          00053 ERR_1   EQU     0               ; Error flag
                    00054
                    00055 ; EEPROM Bits
                    00056
  00000007          00057 DI      EQU     7               ; EEPROM input
  00000006          00058 DO      EQU     6               ; EEPROM output
                    00059
                    00060 ; I2C Device Bits
                    00061
  00000004          00062 SDA     EQU     4               ; RB7, data in/out
  00000003          00063 SCL     EQU     3               ; RB6, serial clock
                    00064
                    00065 ;End of files/bits equate
                    00066
0000                00067         ORG     00h             ; Reset Vector
0000 2810           00068         goto    starting
                    00069
0004                00070         ORG     04h             ; Interrupt Vector
0004 28EB           00071         goto    service_int
                    00072
0010                00073         ORG     10h             ; Begining of Program space
0010                00074 starting
0010 01AE           00075         clrf    DISPVAL         ; Blank out LEDs
0011 01A0           00076         clrf    FLAG            ; Clear error register
0012 0187           00077         clrf    PORTC           ; Set SDA, SCL low when not tri-stated
0013 3008           00078         movlw   08h
0014 00AB           00079         movwf   ROW             ; Set initial row to be strobed
0015 303E           00080         movlw   B'00111110'     ; I2C 7 bit slave mode with master
0016 0094           00081         movwf   SSPCON          ; mode enabled
0017 1683           00082         bsf     STATUS,RP0      ; Select Bank1
0018 0186           00083         clrf    TRISB           ; Set PORT_B to all outputs
0019 3008           00084         movlw   B'00001000'     ; Enable SSP interrupt
001A 008C           00085         movwf   PIE1
001B 30A2           00086         movlw   b'10100010'     ; Slave address
001C 0093           00087         movwf   SSPADD
001D 1587           00088         bsf     TRISC, 3        ; Set SCL high
001E 1607           00089         bsf     TRISC, 4        ; Set SDA high
001F 1283           00090         bcf     STATUS,RP0      ; Select Bank0
0020 118C           00091         bcf     PIR1,3          ; Clear SSP interrupt flag
0021 30C0           00092         movlw   B'11000000'     ; Enable interrupts
0022 008B           00093         movwf   INTCON
                    00094
0023                00095 KbdWait
0023 01AA           00096         clrf    TEMP
0024 216B           00097         call    SetupDelay
0025 2102           00098         call    ScanKbd         ; Check for key pressed
0026 1283           00099         bcf     STATUS,RP0
0027 082C           00100         movf    NEW_KEY,W       ; Get latest key
0028 39FF           00101         andlw   0FFh            ; Key pressed?
0029 1903           00102         btfsc   STATUS,Z
002A 2823           00103         goto    KbdWait         ; No, go check again
002B 00A5           00104         movwf   DATAO           ; Yes, output it on I2C bus
002C 30A6           00105         movlw   B'10100110'     ; Address of device being addressed
002D 00A6           00106         movwf   SLAVE
002E 1683           00107         bsf     STATUS,RP0
002F                00108 CheckAgain
002F 1A14           00109         btfsc   SSPSTAT,4       ;If STOP bit received last...
0030 2833           00110         goto    Goxmit          ;  OK to transmit
0031 1994           00111         btfsc   SSPSTAT,3       ;If START bit recieved last...
0032 282F           00112         goto    CheckAgain      ;  wait for STOP bit
0033                00113 Goxmit
0033 1283           00114         bcf     STATUS,RP0
0034 138B           00115         bcf     INTCON,7        ; Disable interrupts
0035 203F           00116         call    WRBYTE          ; Output byte
0036 1C20           00117         btfss   FLAG,ERR_1      ; Check for error
```


© 1997 Microchip Technology Inc.

```
0037 283D          00118         goto    Checkout        ; No error, go on
0038 0822          00119         movf    ERCODE,W        ; Get error code
0039 0086          00120         movwf   PORTB           ; Put error code on LEDs
003A 00AE          00121         movwf   DISPVAL
003B 1020          00122         bcf     FLAG,ERR_1      ; Clear error flag
003C 282F          00123         goto    CheckAgain
003D               00124 Checkout
003D 178B          00125         bsf     INTCON,7        ; Enable interrupts
003E 2823          00126         goto    KbdWait
                   00127
                   00128 ;-------------------------------------------------------------------
                   00129 ;       BYTE-WRITE, write one byte to I2C (Master Mode)
                   00130 ;-------------------------------------------------------------------
                   00131 ;       Input   :       DATAO   = data to be written
                   00132 ;                       ADDR    = destination address
                   00133 ;                       SLAVE   = device address (1010xxx0)
                   00134 ;       Output  :       Data written to EEPROM device
                   00135 ;-------------------------------------------------------------------
                   00136 ;
003F               00137 WRBYTE
003F 1283          00138         bcf     STATUS,RP0
0040 0826          00139         movf    SLAVE,W         ; Send SLAVE address
0041 00A7          00140         movwf   TXBUF           ;  to TX buffer
0042 20BF          00141         call    BSTART          ; Generate START bit
0043 2071          00142         call    TX              ; Output SLAVE data address
0044 1283          00143         bcf     STATUS,RP0
0045 0825          00144         movf    DATAO,W         ; Move DATA
0046 00A7          00145         movwf   TXBUF           ; into transmit buffer
0047 2071          00146         call    TX              ; Output DATA and detect acknowledgement
0048 20CD          00147         call    BSTOP           ; Generate STOP bit
0049 0008          00148         return
                   00149
                   00150 ;-------------------------------------------------------------------
                   00151 ;       BYTE-READ, read one byte from I2C (Master Mode)
                   00152 ;-------------------------------------------------------------------
                   00153 ;       Input   :       ADDR    = source address
                   00154 ;                       SLAVE   = device address (1010xxx0)
                   00155 ;       Output  :       DATAI   = data read from serial EEPROM
                   00156 ;-------------------------------------------------------------------
                   00157
004A               00158 RDBYTE
004A 1283          00159         bcf     STATUS,RP0
004B 0826          00160         movf    SLAVE,W         ; Move SLAVE address
004C 00A7          00161         movwf   TXBUF           ;  into buffer (R/W = 0)
004D 20BF          00162         call    BSTART          ; Generate START bit
004E 2071          00163         call    TX              ; Output SLAVE address. Check ACK.
004F 1283          00164         bcf     STATUS,RP0
0050 0823          00165         movf    ADDR,W          ; Put slave data address into
0051 00A7          00166         movwf   TXBUF           ;  Xmit buffer
0052 2071          00167         call    TX              ; Output WORD address. Check ACK.
0053 20BF          00168         call    BSTART          ; START READ
0054 1283          00169         bcf     STATUS,RP0
0055 0826          00170         movf    SLAVE,W         ;
0056 00A7          00171         movwf   TXBUF
0057 1427          00172         bsf     TXBUF,0         ; Specify READ mode (R/W = 1)
0058 2071          00173         call    TX              ; Output SLAVE address
0059 205F          00174         call    RX              ; READ in data and acknowledge
005A 20CD          00175         call    BSTOP           ; Generate STOP bit
005B 1283          00176         bcf     STATUS,RP0
005C 0828          00177         movf    RXBUF,W         ; Save data from buffer
005D 00A4          00178         movwf   DATAI           ; to DATAI file.
005E 0008          00179         return
                   00180
                   00181 ;-------------------------------------------------------------------
                   00182 ;       RECEIVE eight data bits subroutine
                   00183 ;-------------------------------------------------------------------
```


© 1997 Microchip Technology Inc.

```
                        00184 ;       Input   :       None
                        00185 ;       Output  :       RXBUF = 8-bit data received
                        00186 ;-------------------------------------------------------------------
                        00187
005F                    00188 RX
005F 1283               00189         bcf     STATUS,RP0
0060 3008               00190         movlw   .8              ; 8 bits of data
0061 00A9               00191         movwf   COUNT
0062 01A8               00192         clrf    RXBUF
                        00193 ;
0063                    00194 RXLP
0063 0DA8               00195         rlf     RXBUF, F        ; Shift data to buffer
0064 1C03               00196         btfss   3,0
0065 1028               00197         bcf     RXBUF,0         ; carry ---> f(0)
0066 1803               00198         btfsc   3,0
0067 1428               00199         bsf     RXBUF,0
0068 2087               00200         call    BITIN
0069 1283               00201         bcf     STATUS,RP0
006A 1BA1               00202         btfsc   EEPROM,DI
006B 1428               00203         bsf     RXBUF,0         ; Input bit =1
006C 0BA9               00204         decfsz  COUNT, F        ; 8 bits?
006D 2863               00205         goto    RXLP
006E 1721               00206         bsf     EEPROM,DO       ; Set acknowledge bit = 1
006F 209E               00207         call    BITOUT          ; to STOP further input
0070 3400               00208         retlw   0
                        00209
                        00210 ;-------------------------------------------------------------------
                        00211 ;       TRANSMIT 8 data bits subroutine
                        00212 ;-------------------------------------------------------------------
                        00213 ;       Input   :       TXBUF
                        00214 ;       Output  :       Data X'mitted to EEPROM device
                        00215 ;-------------------------------------------------------------------
                        00216
0071                    00217 TX
0071 1283               00218         bcf     STATUS,RP0
0072 3008               00219         movlw   .8
0073 00A9               00220         movwf   COUNT
                        00221 ;
0074                    00222 TXLP
0074 1321               00223         bcf     EEPROM,DO       ; Shift data bit out.
0075 1BA7               00224         btfsc   TXBUF,7         ; If shifted bit = 0, data bit = 0
0076 1721               00225         bsf     EEPROM,DO       ; Otherwise data bit = 1
0077 209E               00226         call    BITOUT          ; Serial data out
0078 1283               00227         bcf     STATUS,RP0
0079 0DA7               00228         rlf     TXBUF, F        ; Rotate TXBUF left
007A 1C03               00229         btfss   3,0             ; f(6) ---> f(7)
007B 1027               00230         bcf     TXBUF,0         ; f(7) ---> carry
007C 1803               00231         btfsc   3,0             ; carry ---> f(0)
007D 1427               00232         bsf     TXBUF,0
007E 0BA9               00233         decfsz  COUNT, F        ; 8 bits done?
007F 2874               00234         goto    TXLP            ; No.
0080 2087               00235         call    BITIN           ; Read acknowledge bit
0081 1283               00236         bcf     STATUS,RP0
0082 3003               00237         movlw   3
0083 1BA1               00238         btfsc   EEPROM,DI       ; Check for acknowledgement
0084 20DE               00239         call    ERR             ; No acknowledge from device
0085 1283               00240         bcf     STATUS,RP0
0086 3400               00241         retlw   0
                        00242
                        00243 ;-------------------------------------------------------------------
                        00244 ;       Single bit receive from I2C to PIC
                        00245 ;-------------------------------------------------------------------
                        00246 ;       Input   :       None
                        00247 ;       Output  :       Data bit received
                        00248 ;-------------------------------------------------------------------
                        00249
```


© 1997 Microchip Technology Inc.

```
0087                00250 BITIN
0087 1683           00251         bsf     STATUS,RP0
0088 1607           00252         bsf     TRISC,SDA       ; Set SDA for input
0089 1283           00253         bcf     STATUS,RP0
008A 13A1           00254         bcf     EEPROM,DI
008B 1683           00255         bsf     STATUS,RP0
008C 1587           00256         bsf     TRISC,SCL       ; Clock high
008D 3001           00257         movlw   1
008E 1283           00258         bcf     STATUS,RP0
008F 1987           00259         btfsc   PORTC,SCL       ; Skip if SCL  is high
0090 2895           00260         goto    BIT1
0091 1283           00261         bcf     STATUS,RP0
0092 1C20           00262         btfss   FLAG,ERR_1      ; Remain as first error encountered
0093 00A2           00263         movwf   ERCODE          ; Save error code
0094 1420           00264         bsf     FLAG,ERR_1      ; Set error flag
0095                00265 BIT1
0095 1283           00266         bcf     STATUS,RP0
0096 1E07           00267         btfss   PORTC,SDA       ; Read SDA pin, for ACK low
0097 289A           00268         goto    ACKOK
0098 1283           00269         bcf     STATUS,RP0
0099 17A1           00270         bsf     EEPROM,DI       ; DI = 1
009A                00271 ACKOK
009A 1683           00272         bsf     STATUS,RP0
009B 0000           00273         nop                     ; Delay
009C 1187           00274         bcf     TRISC,SCL       ; Return SCL to low
009D 3400           00275         retlw   0
                    00276
                    00277 ;-------------------------------------------------------------------
                    00278 ;       Single bit data transmit from PIC to I2C
                    00279 ;-------------------------------------------------------------------
                    00280 ;       Input   :       EEPROM register, bit DO
                    00281 ;       Output  :       Bit transmitted over I2C
                    00282 ;                       Error bits set as necessary
                    00283 ;-------------------------------------------------------------------
                    00284
009E                00285 BITOUT
009E 1283           00286         bcf     STATUS,RP0
009F 1F21           00287         btfss   EEPROM,DO
00A0 28AB           00288         goto    BIT0
00A1 1683           00289         bsf     STATUS,RP0
00A2 1607           00290         bsf     TRISC,SDA       ; Output bit 0
00A3 3002           00291         movlw   2
00A4 1283           00292         bcf     STATUS,RP0
00A5 1A07           00293         btfsc   PORTC,SDA       ; Check for error code 2
00A6 28B0           00294         goto    CLK1
00A7 1C20           00295         btfss   FLAG,ERR_1      ; Remain as first error encountered
00A8 00A2           00296         movwf   ERCODE          ; Save error code
00A9 1420           00297         bsf     FLAG,ERR_1      ; Set error flag
00AA 28B0           00298         goto    CLK1            ; SDA locked low by device
                    00299 ;
00AB                00300 BIT0
00AB 1683           00301         bsf     STATUS,RP0
00AC 1207           00302         bcf     TRISC,SDA       ; Output bit 0
00AD 0000           00303         nop                      ; Delay
00AE 0000           00304         nop
00AF 0000           00305         nop
00B0                00306 CLK1
00B0 1683           00307         bsf     STATUS,RP0
00B1 1587           00308         bsf     TRISC,SCL
00B2 3001           00309         movlw   1               ; Error code 1
00B3 1283           00310         bcf     STATUS,RP0
00B4 1987           00311         btfsc   PORTC,SCL       ; SCL locked low?
00B5 28BA           00312         goto    BIT2            ; No.
00B6 1283           00313         bcf     STATUS,RP0
00B7 1C20           00314         btfss   FLAG,ERR_1      ; Yes.
00B8 00A2           00315         movwf   ERCODE          ; Save error code
```


© 1997 Microchip Technology Inc.

```
00B9 1420          00316         bsf     FLAG,ERR_1      ; Set error flag
00BA               00317 BIT2
00BA 0000          00318         nop
00BB 0000          00319         nop
00BC 1683          00320         bsf     STATUS,RP0
00BD 1187          00321         bcf     TRISC,SCL       ; Return SCL to low
00BE 3400          00322         retlw   0
                   00323
                   00324 ;-------------------------------------------------------------------
                   00325 ;       START bit generation routine
                   00326 ;-------------------------------------------------------------------
                   00327 ;       input   : none
                   00328 ;       output  : initialize bus communication
                   00329 ;-------------------------------------------------------------------
                   00330
                   00331 ;Generate START bit (SCL is high while SDA goes from high to low
                   00332 ;transition) and check status of the serial clock.
00BF               00333 BSTART
00BF 1683          00334         bsf     STATUS,RP0
00C0 1607          00335         bsf     TRISC,SDA       ; Make sure SDA is high
00C1 1587          00336         bsf     TRISC,SCL       ; Set clock high
00C2 3001          00337         movlw   1               ; Ready error status code 1
00C3 1283          00338         bcf     STATUS,RP0
00C4 1D87          00339         btfss   PORTC,SCL       ; Locked?
00C5 20DE          00340         call    ERR             ; SCL locked low by device, flag error
00C6 1683          00341         bsf     STATUS,RP0
00C7 1207          00342         bcf     TRISC,SDA       ; SDA goes low during SCL high
00C8 0000          00343         nop                     ; Timing adjustment, 1.5uS @2MHz
00C9 0000          00344         nop
00CA 0000          00345         nop
00CB 1187          00346         bcf     TRISC,SCL       ; Start clock train
00CC 3400          00347         retlw   0
                   00348
                   00349 ;-------------------------------------------------------------------
                   00350 ;       STOP bit generation routine
                   00351 ;-------------------------------------------------------------------
                   00352 ;       Input   :       None
                   00353 ;       Output  :       Bus communication, STOP condition
                   00354 ;-------------------------------------------------------------------
                   00355
                   00356 ;Generate STOP bit (SDA goes from low to high during SCL high state)
                   00357 ;and check bus conditions.
                   00358
00CD               00359 BSTOP
00CD 1683          00360         bsf     STATUS,RP0
00CE 1207          00361         bcf     TRISC,SDA       ; Return SDA to low
00CF 1587          00362         bsf     TRISC,SCL       ; Set SCL high
00D0 0000          00363         nop
00D1 0000          00364         nop
00D2 0000          00365         nop
00D3 3001          00366         movlw   1               ; Ready error code 1
00D4 1283          00367         bcf     STATUS,RP0
00D5 1D87          00368         btfss   PORTC,SCL       ; High?
00D6 20DE          00369         call    ERR             ; No, SCL locked low by device
00D7 1683          00370         bsf     STATUS,RP0
00D8 1607          00371         bsf     TRISC,SDA       ; SDA goes low to high during SCL high
00D9 3004          00372         movlw   4               ; Ready error code 4
00DA 1E07          00373         btfss   TRISC,SDA       ; High?
00DB 20DE          00374         call    ERR             ; No, SDA bus not release for STOP
00DC 1283          00375         bcf     STATUS,RP0
00DD 3400          00376         retlw   0
                   00377
                   00378 ;-------------------------------------------------------------------
                   00379 ;    Two wire/I2C - CPU communication error status table and subroutine
                   00380 ;-------------------------------------------------------------------
                   00381 ;  input  :     W-reg   = error code
```


© 1997 Microchip Technology Inc.

```
                        00382 ;  output :     ERCODE  = error code
                        00383 ;           FLAG(ERR_1) = 1
                        00384 ;
                        00385 ;         code                    error status mode
                        00386 ;        ------      ----------------------------------------------
                        00387 ;         1   :    SCL locked low by device (bus is still busy)
                        00388 ;         2   :    SDA locked low by device (bus is still busy)
                        00389 ;         3   :    No acknowledge from device (no handshake)
                        00390 ;         4   :    SDA bus not released for master to generate STOP bit
                        00391 ;-------------------------------------------------------------------
                        00392 ;
                        00393 ;Subroutine to identify the status of the serial clock (SCL) and serial
                        00394 ;(data SDA) condition according to the error status table.Codes
                        00395 ;generated are useful for bus/device diagnosis.
                        00396
00DE                    00397 ERR
00DE 1283               00398         bcf     STATUS,RP0
00DF 1C20               00399         btfss   FLAG,ERR_1      ; Keep first error reported
00E0 00A2               00400         movwf   ERCODE          ; Save error code
00E1 1420               00401         bsf     FLAG,ERR_1      ; Set error flag
00E2 3400               00402         retlw   0
                        00403
                        00404 ;-------------------------------------------------------------------
                        00405 ;       DELAY, Provide a 1.54mS delay
                        00406 ;-------------------------------------------------------------------
                        00407 ;       Input   :       None
                        00408 ;       Output  :       None
                        00409 ;-------------------------------------------------------------------
                        00410
00E3                    00411 delay
00E3 1283               00412         bcf     STATUS,RP0
00E4 01AA               00413         clrf    TEMP            ;clear last location
00E5                    00414 dly1
00E5 0000               00415         nop
00E6 0000               00416         nop
00E7 0000               00417         nop
00E8 0BAA               00418         decfsz  TEMP, F         ;reduce count
00E9 28E5               00419         goto    dly1            ;loop
00EA 3400               00420         retlw   0
                        00421
                        00422 ;-------------------------------------------------------------------
                        00423 ; Interrupt service routine
                        00424 ;  Only the SSP interrupt is enabled.  This routine will read the I2C
                        00425 ;  data and output it on the LEDs.
                        00426 ;-------------------------------------------------------------------
                        00427
00EB                    00428 service_int
00EB 118C               00429         bcf     PIR1,3          ; Clear SSP interrupt
00EC 00B0               00430         movwf   TEMP_W          ; Save W register
00ED 0E03               00431         swapf   STATUS,W        ; Get STATUS register
00EE 00B1               00432         movwf   TEMP_STAT       ; Save STATUS register
00EF 1683               00433         bsf     STATUS,RP0
00F0 1C14               00434         btfss   SSPSTAT,0       ; Check Buffer Full Flag
00F1 28FA               00435         goto    IntOut          ; No data received, so exit
00F2 1283               00436         bcf     STATUS,RP0
00F3 0813               00437         movf    SSPBUF,W        ; Get I2C data
00F4 1683               00438         bsf     STATUS,RP0
00F5 1E94               00439         btfss   SSPSTAT,5       ; If Address received last...
00F6 28FA               00440         goto    IntOut          ;  exit without saving it
00F7 1283               00441         bcf     STATUS,RP0
00F8 0086               00442         movwf   PORTB           ; Display received data on LEDs
00F9 00AE               00443         movwf   DISPVAL
00FA                    00444 IntOut
00FA 1683               00445         bsf     STATUS,RP0
00FB 158C               00446         bsf     PIE1,3          ; Re-enable SSP interrupt
00FC 1283               00447         bcf     STATUS,RP0
```


© 1997 Microchip Technology Inc.

```
00FD 0E31          00448         swapf   TEMP_STAT,W     ; Restore STATUS register
00FE 0083          00449         movwf   STATUS
00FF 0EB0          00450         swapf   TEMP_W,1
0100 0E30          00451         swapf   TEMP_W,W        ; Restore W register
0101 0009          00452         retfie
                   00453
                   00454 ;-----------------------------------------------------------------
                   00455 ; Keyboard scan routine
                   00456 ;  This routine scans the keypad connected to PORT_B, and
                   00457 ;  returns the pressed key in New_Key.
                   00458 ;-----------------------------------------------------------------
                   00459
0102               00460 ScanKbd
0102 1283          00461         bcf     STATUS,RP0
0103 01AC          00462         clrf    NEW_KEY         ; Clear key register
0104 082D          00463         movf    OLD_KEY,w       ; If key was pressed last pass through
0105 1D03          00464         btfss   STATUS,Z        ;  goto Debounce
0106 291C          00465         goto    Debounce
0107               00466 Kbdloop
0107 1003          00467         bcf     STATUS,C
0108 0DAB          00468         rlf     ROW, F          ; Select next row to strobe
0109 1C03          00469         btfss   STATUS,C
010A 290D          00470         goto    Notdone
010B 3010          00471         movlw   010h            ; Start over at first row
010C 00AB          00472         movwf   ROW
010D               00473 Notdone
010D 0186          00474         clrf    PORTB
010E 1683          00475         bsf     STATUS,RP0
010F 300F          00476         movlw   00Fh            ; Set PORT_B for keypad read
0110 0086          00477         movwf   TRISB
0111 1283          00478         bcf     STATUS,RP0
0112 082B          00479         movf    ROW,W           ; Output Row
0113 0086          00480         movwf   PORTB           ; Read colums
0114 300F          00481         movlw   0Fh             ; Mask out rows
0115 0506          00482         andwf   PORTB,W         ; Check for Key press
0116 1903          00483         btfsc   STATUS,Z
0117 2933          00484         goto    KBDOUT          ; No key pressed, exit
0118 0186          00485         clrf    PORTB
0119 042B          00486         iorwf   ROW,W           ; Key pressed, save it
011A 00AC          00487         movwf   NEW_KEY
011B 00AD          00488         movwf   OLD_KEY
011C               00489 Debounce
011C 0186          00490         clrf    PORTB
011D 1683          00491         bsf     STATUS,RP0
011E 0186          00492         clrf    TRISB
011F 1283          00493         bcf     STATUS,RP0
0120 082E          00494         movf    DISPVAL,W       ; Set LEDs
0121 0086          00495         movwf   PORTB
0122 01AA          00496         clrf    TEMP
0123 216B          00497         call    SetupDelay
0124 216B          00498         call    SetupDelay      ; Delay for key debounce
0125 0186          00499         clrf    PORTB
0126 1683          00500         bsf     STATUS,RP0
0127 300F          00501         movlw   0Fh             ; Set PORT_B for keypad read
0128 0086          00502         movwf   TRISB
0129 1283          00503         bcf     STATUS,RP0
012A 082B          00504         movf    ROW,W
012B 0086          00505         movwf   PORTB           ; Output Row
012C 082D          00506         movf    OLD_KEY,W
012D 0606          00507         xorwf   PORTB,w         ; Compare key with last key pressed
012E 1903          00508         btfsc   STATUS,Z
012F 2933          00509         goto    KBDOUT
0130 0186          00510         clrf    PORTB           ; Key released, clear registers
0131 01AC          00511         clrf    NEW_KEY
0132 01AD          00512         clrf    OLD_KEY
0133               00513 KBDOUT
```


© 1997 Microchip Technology Inc.

```
0133 082E           00514          movf    DISPVAL,W       ; Set LEDs
0134 0086           00515          movwf   PORTB
0135 1683           00516          bsf     STATUS,RP0
0136 0186           00517          clrf    TRISB
0137 1283           00518          bcf     STATUS,RP0
0138                00519 KEY_DEC
0138 300F           00520          movlw   00Fh
0139 052C           00521          andwf   NEW_KEY,W       ; Get column of key pressed
013A 00AA           00522          movwf   TEMP
013B 1903           00523          btfsc   STATUS,Z        ; If no key pressed, exit
013C 2955           00524          goto    DECOUT
013D 30FF           00525          movlw   0FFh            ; Initialize the W register
013E                00526 DECL1
013E 3E01           00527          addlw   001h            ; Count column
013F 0CAA           00528          rrf     TEMP, F         ; Rotate column until it is found
0140 1C03           00529          btfss   STATUS,C
0141 293E           00530          goto    DECL1
0142 00AA           00531          movwf   TEMP
0143 0E2C           00532          swapf   NEW_KEY,W       ; Get row of key pressed
0144 390F           00533          andlw   00Fh
0145 1903           00534          btfsc   STATUS,Z
0146 2955           00535          goto    DECOUT          ;If no key pressed, exit
0147 00AF           00536          movwf   TEMP1
0148 30FF           00537          movlw   0FFH
0149                00538 DECL2
0149 3E01           00539          addlw   001h            ; Count row
014A 0CAF           00540          rrf     TEMP1, F        ; Rotate row until it is found
014B 1C03           00541          btfss   STATUS,C
014C 2949           00542          goto    DECL2
014D 00AF           00543          movwf   TEMP1
014E 1003           00544          bcf     STATUS,C
014F 0DAF           00545          rlf     TEMP1, F        ; Move row to upper nibble
0150 0DAF           00546          rlf     TEMP1, F
0151 082F           00547          movf    TEMP1,W
0152 072A           00548          addwf   TEMP,W          ; Add column to row value
0153 2156           00549          call    DEC_TABL        ; Get ASCII value of key
0154 00AC           00550          movwf   NEW_KEY
0155                00551 DECOUT
0155 0008           00552          return
0156                00553 DEC_TABL
0156 00AA           00554          movwf   TEMP            ; Save key value
0157 3001           00555          movlw   01h
0158 008A           00556          movwf   PCLATH          ; Setup for second page of RAM
0159 082A           00557          movf    TEMP,W
015A 0782           00558          addwf   PCL, F          ; Jump, return with ASCII value
015B 3446           00559          retlw   'F'
015C 3442           00560          retlw   'B'
015D 3430           00561          retlw   '0'
015E 3441           00562          retlw   'A'
015F 3445           00563          retlw   'E'
0160 3439           00564          retlw   '9'
0161 3438           00565          retlw   '8'
0162 3437           00566          retlw   '7'
0163 3444           00567          retlw   'D'
0164 3436           00568          retlw   '6'
0165 3435           00569          retlw   '5'
0166 3434           00570          retlw   '4'
0167 3443           00571          retlw   'C'
0168 3433           00572          retlw   '3'
0169 3432           00573          retlw   '2'
016A 3431           00574          retlw   '1'
                    00575
                    00576 ;*****************************************************************
                    00577 ;*This routine is a software delay.                              *
                    00578 ;*At 4Mhz clock, the loop takes 3uS, so initialize TEMP with     *
                    00579 ;*a value of 3 to give 9uS, plus the move etc should result in  *
```


© 1997 Microchip Technology Inc.

```
                    00580 ;*a total time of > 10uS.                                 *
                    00581 ;*********************************************************************
                    00582
016B                00583 SetupDelay
016B 0BAA           00584         decfsz  TEMP, F
016C 296B           00585         goto    SetupDelay
016D 0008           00586         return
                    00587
                    00588         END


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---X----------- XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXX-- ----------------

All other memory blocks unused.

Program Memory Words Used:   352
Program Memory Words Free:  1696


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,    29 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Using the 8-Bit Parallel Slave Port

Author: *Scott Fink*
Microchip Technology Inc.

INTRODUCTION

PIC16C64/74 microcontrollers from Microchip Technology Inc. can be interfaced with ease into a multi-microprocessor environment using its built-in Parallel Slave Port (PSP). With their very high operating speeds (cycle times as low as 200 ns with a clock rate of 20 MHz), and an array of on-chip peripherals, these microcontrollers make ideal smart interfaces to the real world.

IMPLEMENTATION

PORTD operates as an 8-bit wide Parallel Slave Port, with PORTE providing the control signals. In parallel slave mode, PORTD is asynchronously readable and writable by the external world through the chip select (RE2/$\overline{CS}$), Read (RE0/$\overline{RD}$), and Write (RE1/$\overline{WR}$) control inputs.

In order to use the Parallel Slave Port, the data direction bits in the TRISE register corresponding to $\overline{RD}$, $\overline{WR}$, and $\overline{CS}$ (TRISE<2:0>) must be configured as inputs (set = 1) and control bit PSPMODE (TRISE<4>) must be set.

The port pins are connected to two 8-bit latches, one for data output (from the PIC16CXXX) and one for data input. The PIC16CXXX sends data by writing to the output latch, and receives data by reading the input latch (note that the input and output latches are at the same address). In PSP mode the TRISD register is ignored, since the external device connected to the slave port controls the direction of data flow.

When the external device performs either a read or a write operation to the PIC16CXXX, interrupt flag, PSPIF (PIR1<7>), will be set and the processor interrupted if bit PSPIE (PIE1<7>) is set and interrupts are enabled (enable bits GIE and PEIE, (INTCON<7:6>) set). When the interrupt is serviced, bit PSPIF must be cleared by software.

The read-only status flag bit IBF, Input Buffer Full (TRISE<7>), is set if a received word is waiting to be read. Bit IBF is cleared upon read of the input buffer latch. If another word is received prior to the first being read, status flag bit IBOV (TRISE<5>) is set. Bit IBOV can be cleared by software.

The Output Buffer Full status bit, OBF (TRISE<6>), is set if a word written to PORTD latch is waiting to be read by the external bus.

When not in Parallel Slave Port mode the IBF and OBF bits are cleared. If flag bit IBOV was previously set, however, it must be cleared by software.

Note that the following registers are for a PIC16C74 and not all peripherals are available on the PIC16C64.

TABLE 1: SUMMARY OF PARALLEL SLAVE PORT REGISTERS

Register Name	Function	Address	Power-on Reset Value
PORTD	Parallel slave port Read/Write Data	08h	xxxx xxxx
TRISD	PORTD data direction register	88h	1111 1111
PORTE	Read/Write/Chip Select signals	09h	---- -xxx
TRISE	Control bits for PORTD slave port	89h	0000 -111
INTCON	peripheral and global interrupt enable bits	0Bh	0000 000x
PIR1	Interrupt register (PSPIF bit)	0Ch	0000 0000
PIE1	Interrupt Enable register (PSPIE bit)	8Ch	0000 0000

© 1997 Microchip Technology Inc.

TABLE 2: PORTE FUNCTIONS

Name	Bit#	Buffer Type	Function
RE0/$\overline{RD}$/AN5	bit0	ST/TTL(1)	Input/output port pin or read control input in parallel slave port mode or analog input: $\overline{RD}$ 1 = Not a read operation 0 = Read operation. Reads PORTD register (if chip selected)
RE1/$\overline{WR}$/AN6	bit1	ST/TTL(1)	Input/output port pin or write control input in parallel slave port mode or analog input: $\overline{WR}$ 1 = Not a write operation 0 = Write operation. Writes PORTD register (if chip selected)
RE2/$\overline{CS}$/AN7	bit2	ST/TTL(1)	Input/output port pin or chip select control input in parallel slave port mode or analog input: $\overline{CS}$ 1 = Device is not selected 0 = Device is selected

Legend: ST = Schmitt Trigger input TTL = TTL input
Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port Mode.

FIGURE 1: TRISE REGISTER

R-0	R-0	R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1
IBF	OBF	IBOV	PSPMODE	—	Bit2	Bit1	Bit0
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **IBF:** Input Buffer Full Status bit
1 = A word has been received and waiting to be read by the CPU
0 = No word has been received

bit 6: **OBF:** Output Buffer Full Status bit
1 = The output buffer still holds a previously written word
0 = The output buffer has been read

bit 5: **IBOV:** Input Buffer Overflow Detect bit (in microprocessor mode)
1 = A write occurred when a previously input word has not been read (must be cleared in software)
0 = No overflow occurred

bit 4: **PSPMODE:** Parallel Slave Port Mode Select bit
1 = Parallel slave port mode
0 = General purpose I/O mode

bit 3: **Unimplemented:** Read as '0'

bit 2: **Bit2:** Direction control bit for pin RE2/$\overline{CS}$/AN7
1 = Input
0 = Output

bit 1: **Bit1:** Direction control bit for pin RE1/$\overline{WR}$/AN6
1 = Input
0 = Output

bit 0: **Bit0:** Direction control bit for pin RE0/$\overline{RD}$/AN5
1 = Input
0 = Output

© 1997 Microchip Technology Inc.

FIGURE 2: PIE1 REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **PSPIE:** Parallel Slave Port Read/Write Interrupt Enable bit
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

bit 6: **ADIE:** A/D Converter Interrupt Enable bit
1 = Enables the A/D interrupt
0 = Disables the A/D interrupt

bit 5: **RCIE:** USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4: **TXIE:** USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3: **SSPIE:** Synchronous Serial Port Interrupt Enable bit
1 = Enables the SSP interrupt
0 = Disables the SSP interrupt

bit 2: **CCP1IE:** CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1: **TMR2IE:** TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0: **TMR1IE:** TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

© 1997 Microchip Technology Inc.

FIGURE 3: PIR1 REGISTER

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **PSPIF(1):** Parallel Slave Port Read/Write Interrupt Flag bit
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred

bit 6: **ADIF**: A/D Converter Interrupt Flag bit
1 = An A/D conversion completed (must be cleared in software)
0 = The A/D conversion is not complete

bit 5: **RCIF**: USART Receive Interrupt Flag bit
1 = The USART receive buffer is full (cleared by reading RCREG)
0 = The USART receive buffer is empty

bit 4: **TXIF**: USART Transmit Interrupt Flag bit
1 = The USART transmit buffer is empty (cleared by writing to TXREG)
0 = The USART transmit buffer is full

bit 3: **SSPIF**: Synchronous Serial Port Interrupt Flag bit
1 = The transmission/reception is complete (must be cleared in software)
0 = Waiting to transmit/receive

bit 2: **CCP1IF**: CCP1 Interrupt Flag bit
Capture Mode
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare Mode
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM Mode
Unused in this mode

bit 1: **TMR2IF**: TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit 0: **TMR1IF**: TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

Note 1: PIC16C73/73A/76 devices do not have a Parallel Slave Port implemented, this bit location is reserved on these devices, always maintain this bit clear.

Interrupt flag bits get set when an interrupt condition occurs regardless of the state of its corresponding enable bit or the global enable bit, GIE (INTCON<7>). User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt.

© 1997 Microchip Technology Inc.

TABLE 3: INTCON REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n = Value at POR reset

bit 7: **GIE:** Global Interrupt Enable bit
1 = Enables all un-masked interrupts
0 = Disables all interrupts

bit 6: **PEIE**: Peripheral Interrupt Enable bit
1 = Enables all un-masked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5: **T0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4: **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3: **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2: **T0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1: **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

bit 0: **RBIF**: RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

3

© 1997 Microchip Technology Inc

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: PIC16C64/74 PARALLEL SLAVE PORT

```
MPASM 01.40 Released            PSP64.ASM   1-16-1997  17:03:44         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;*******************************************************************
                     00002 ;* 16C64/74 Parallel Slave port
                     00003 ;*  This program demonstrates the Parallel Slave Port function of
                     00004 ;*  the PIC16C64/74.  The program is interrupt driven, when the PIC
                     00005 ;*  is either read from or written to, an interrupt is generated. If
                     00006 ;*  the interrupt was caused by a read, a register is incremented, and
                     00007 ;*  the new count is placed in an output queue.  If the interrupt was
                     00008 ;*  caused by a write, the data is put on the Port B pins
                     00009 ;
                     00010 ;     Program:          PSP64.ASM
                     00011 ;     Revision Date:
                     00012 ;                       1-15-97      Compatibility with MPASMWIN 1.40
                     00013 ;
                     00014 ;;*******************************************************************
                     00015         list p=16c64
                     00016         ERRORLEVEL  -302
                     00017 ;
                     00018         include "p16c64.inc"
                     00001         LIST
                     00002 ; P16C64.INC Standard Header File, Ver. 1.01 Microchip Technology, Inc.
                     00238         LIST
                     00019
                     00020 ;Register definitions
  00000020           00021 FLAGREG equ     20h             ;Flag bit register
  00000021           00022 OUTDATA equ     21h             ;Output data
  00000022           00023 INDATA  equ     22h             ;Input data
  00000023           00024 COUNT   equ     23h             ;Count of times output register read
                     00025
                     00026 ;Bit definitions for flag register
  00000000           00027 err     equ     00h             ;Error flag bit
  00000001           00028 OUTRDY  equ     01h             ;Output data ready flag
  00000002           00029 INFULL  equ     02h             ;Input data received flag
                     00030
0000                 00031         org     0000h           ;Reset Vector
0000 2806            00032         goto    Start
                     00033
0005                 00034         org     0005h           ;Interrupt Vector
0005 2820            00035         goto    Service_Int
                     00036
0006                 00037 Start
0006 01A1            00038         clrf    OUTDATA         ;Clear data registers
0007 01A2            00039         clrf    INDATA
0008 1683            00040         bsf     STATUS,RP0      ;Select register Bank1
0009 3017            00041         movlw   b'00010111'     ;Set RD, WR, and CS as
000A 0089            00042         movwf   TRISE           ; inputs, Enable Parallel Slave port
000B 30FF            00043         movlw   0FFh
000C 0086            00044         movwf   TRISB           ;Set Port_B to all outputs
000D 3080            00045         movlw   b'10000000'     ;
000E 008C            00046         movwf   PIE1            ;Enable Parallel Slave Port interrupt
000F 1283            00047         bcf     STATUS,RP0      ;Select register Bank0
                     00048
0010 0821            00049         movf    OUTDATA,W       ;Set output Data in PORTD
0011 0088            00050         movwf   PORTD
0012 30C0            00051         movlw   b'11000000'     ;Set GIE, PEIE (enable interrupts)
```


© 1997 Microchip Technology Inc.

```
0013 008B          00052         movwf   INTCON
                   00053
0014               00054 Loop
0014 1920          00055         btfsc   FLAGREG,INFULL  ;Check if input data received
0015 2819          00056         goto    Checkout        ;No data ready, check output
0016 1120          00057         bcf     FLAGREG,INFULL  ;Clear input data ready flag
0017 0822          00058         movf    INDATA,W        ;Get Input data
0018 0086          00059         movwf   PORTB           ;Output input data to Port_B
0019               00060 Checkout
0019 18A0          00061         btfsc   FLAGREG,OUTRDY  ;Check if data output already
001A 2814          00062         goto    Loop            ;Not output yet, loop
001B 0AA3          00063         incf    COUNT, F        ;Increment output data
001C 0823          00064         movf    COUNT,W         ;Get output data
001D 00A1          00065         movwf   OUTDATA         ;Put data in output queue
001E 14A0          00066         bsf     FLAGREG,OUTRDY  ;Set flag for interrupt routine
001F 2814          00067         goto    Loop
                   00068
                   00069 ;*****************************************************************
                   00070 ;*Interrupt Service Routine
                   00071 ;*       Inputs: FLAGREG - Flag register to/from the main routine:
                   00072 ;*                         Bit 1: OUTRDY - To Service_Int, indicates
                   00073 ;*                                         data ready in output queue
                   00074 ;*               OUTDATA - Output data queue
                   00075 ;*               PIR1    - Interrupt flag register
                   00076 ;*               TRISE   - Parallel slave port flag register
                   00077 ;*               PORTD   - Input data from slave port
                   00078 ;*
                   00079 ;*       Outputs:
                   00080 ;*               PORTD   - Output data to slave port
                   00081 ;*               INDATA  - Input data queue
                   00082 ;*               FLAGREG - Flag register to/from the main routine:
                   00083 ;*                         Bit 0: ERROR -  From Service_Int, indicates
                   00084 ;*                                         input buffer overflow
                   00085 ;*                         Bit 2: INFULL - From Service_Int, indicates
                   00086 ;*                                         data received and in INDATA
                   00087 ;*****************************************************************
                   00088
0020               00089 Service_Int
0020 1F8C          00090         btfss   PIR1,PSPIF      ;Test for Peripheral interrupt
0021 2832          00091         goto    Intout          ;Not a Peripheral interrupt, exit
0022 138C          00092         bcf     PIR1,PSPIF      ;Clear Peripheral interrupt
0023 1683          00093         bsf     STATUS,RP0      ;Select Bank1
0024 1F89          00094         btfss   TRISE,IBF       ;Check if input data ready
0025 282A          00095         goto    Notinput        ;No input, check output
0026 1283          00096         bcf     STATUS,RP0      ;Input ready, select Bank0
0027 1520          00097         bsf     FLAGREG,INFULL  ;Set flag for main routine
0028 0808          00098         movf    PORTD,W         ;Get input data
0029 00A2          00099         movwf   INDATA          ;Put byte in input queue
002A               00100 Notinput
002A 1B09          00101         btfsc   TRISE,OBF       ;Check if output data read
002B 2832          00102         goto    Intout          ;Not read, exit
002C 1283          00103         bcf     STATUS,RP0      ;Select Bank0
002D 1CA0          00104         btfss   FLAGREG,OUTRDY  ;Check if data in output queue
002E 2832          00105         goto    Intout          ;Output not read, exit
002F 0821          00106         movf    OUTDATA,W       ;Get data from queue
0030 0888          00107         movf    PORTD, F        ;Put data in output buffer
0031 10A0          00108         bcf     FLAGREG,OUTRDY  ;Clear flag for main routine
0032               00109 Intout
0032 1683          00110         bsf     STATUS,RP0      ;Select Bank1
0033 1A89          00111         btfsc   TRISE,IBOV      ;Check input buffer overflow flag
0034 2837          00112         goto    Interror        ;If not clear, error
0035 1283          00113         bcf     STATUS,RP0      ;Select Bank0
0036 0009          00114         retfie                  ;Re-enable GIE and return
0037               00115 Interror
0037 1283          00116         bcf     STATUS,RP0      ;Select Bank0
0038 1420          00117         bsf     FLAGREG,err     ;Set error flag for main routine
```

```
0039 0009           00118         retfie                  ;Re-enable GIE and return
                    00119
                    00120         end
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X----XXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXX------

All other memory blocks unused.

Program Memory Words Used:    54
Program Memory Words Free:  1994


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     6 suppressed
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN580

Using Timer1 in Asynchronous Clock Mode

Author: *Mark Palmer*
Microchip Technology Inc.

INTRODUCTION

This application note discusses the use of the PIC16CXXX Timer1 module as an asynchronous clock. The Timer1 module has it own oscillator circuitry, which allows the timer to keep real-time, even when the device is in SLEEP mode. When the device is in sleep, the oscillator will continue to increment TMR1. An overflow of the TMR1 register causes a TMR1 Overflow Interrupt (if enabled) and will wake the processor from sleep. The interrupt service routine can then perform the desired task.

OVERVIEW

Timer1 is a 16-bit counter with a 2-bit prescaler. Timer1 can be incremented from an internal clock, an external clock, or an external oscillator. Timer1 can be configured to synchronize or not synchronize the external clock sources. Asynchronous operation allows Timer1 to increment when the device is in sleep. Figure 1 is a block diagram of Timer1.

To set up Timer1 for asynchronous operation the Timer1 control register, T1CON, must have the following bits configured:

- TMR1CS set (external clock source)
- T1CKS1:T1CKS0 configured for the desired prescaler
- $\overline{\text{T1SYNC}}$ set (asynchronous operation)
- TMR1ON set (enables Timer1)
- T1OSCEN set, if using an external oscillator

In asynchronous operation, if the clock source is an external clock, it is input on the T1CKI pin. If the clock source is a crystal oscillator, the crystal is connected across the T1OSO and T1OSI pins.

When using Timer1 in Asynchronous mode, the use of an external clock minimizes the operating and sleep currents. This is because the timer's internal oscillator circuitry is disabled. Though the external clock may give the lower device currents, the use of a crystal oscillator may lead to lower system current consumption and system cost.

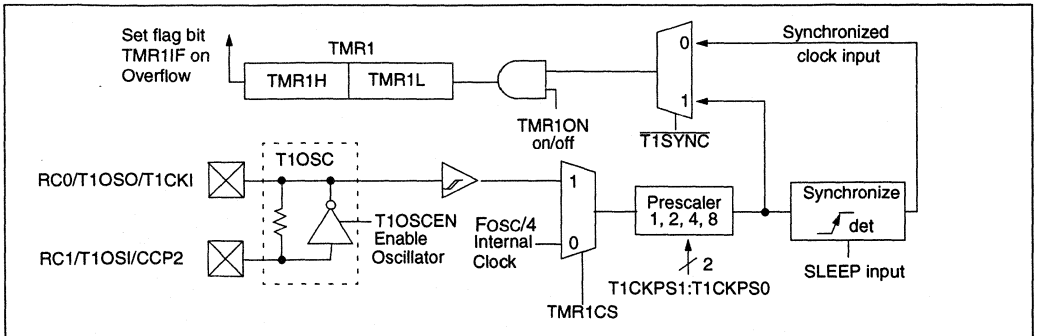
System current consumption can also be reduced by having the TMR1 Overflow Interrupt wake the processor from SLEEP at the desired interval, With a 32.768 kHz crystal, Timer1's overflow rate ranges from 2 to 16 seconds, depending on the prescaler chosen. Table 1 shows Timer1 overflow times for various crystal frequencies and prescaler values.

TABLE 1: TIMER1 OVERFLOW TIMES

	Frequency (kHz)		
Prescale	**32.768**	**100**	**200**
1	2	0.655	0.327
2	4	1.31	0.655
4	8	2.62	1.31
8	16	5.24	2.62

Overflow times in seconds.

FIGURE 1: TIMER1 BLOCK DIAGRAM



When the T1OSCEN bit is cleared, the inverter and feedback resistor are turned off. This eliminates power drain.

© 1997 Microchip Technology Inc.

As can be seen the 32 kHz crystal, gives very nice overflow rates. These crystals, referred to as watch crystals, also can be relatively inexpensive. In many applications the 2 second overflow time, of a 32 kHz crystal, is too long. An easy way to reduce the overflow time is to load the TMR1H register with a value, during the interrupt service routine. Table 2 shows the overflow times, depending on the value loaded into the TMR1H register and a prescale of 1.

TABLE 2: TMR1H LOAD VALUES / TIMER1 OVERFLOW TIMES

TMR1H Load Value	Overflow Time (@ 32.768 kHz)
80h	1 Second
C0h	0.5 Second
E0h	0.25 Second
F0h	0.125 Second

Note: The loading of either TMR1H or TMR1L causes the prescaler to be cleared. When Timer1 is in operation, extreme care should be taken in modifying either the TMR1H or TMR1L registers, since this automatically configures the prescaler to 1.

The code segment shown in Example 1 configures the Timer1 module for asynchronous operation, enables the Timer1 interrupt, and the interrupt service routine loads the TMR1H register with a value.

CONCLUSION

Timer1 gives designers a powerful time-base function. The asynchronous operation and internal oscillator circuitry gives designers the ability to easily keep real-time, while minimizing power consumption and external logic.

© 1997 Microchip Technology Inc.

EXAMPLE 1: TIMER1 CODE SEGMENT FOR ASYNCHRONOUS OPERATION

```
      org   0x000
Reset_V     GOTO    START
;
      org   0x004
PER_INT_V
            BCF     STATUS, RP0     ; Bank0
            BTFSC   PIR1, TMR1IF    ; Timer1 overflowed?
            GOTO    T1_OVRFL        ; YES, Service the Timer1 Overflow Interrupt
;
; Should NEVER get here
;
ERROR1                              ; NO, Unknown Interrupt Source
            BSF     PORTD, 1        ; Toggle a port pin to indicate error
            BCF     PORTD, 1
            GOTO    ERROR1
;
T1_OVRFL
            BCF     PIR1, TMR1IF    ; Clear Timer1 Interrupt Flag
            MOVLW   0x80            ; Since doing key inputs, clear TMR1
            MOVWF   TMR1H           ;   for 1 sec overflow.
            :
            :                       ; Do Interrupt stuff here
            :
            RETFIE                  ; Return / Enable Global Interrupts
;
;
START                               ; POWER_ON Reset (Beginning of program)
            CLRF    STATUS          ; Do initialization (Bank0)
            BCF     T1CON, TMR1ON   ; Timer1 is NOT incrementing
            :
            :                       ; Do Initialization stuff here
            :
            MOVLW   0x80            ; TIM1H:TMR1L = 0x8000 gives 1 second
            MOVWF   TMR1H           ;   overflow, at 32 KHz.
            CLRF    TMR1L           ;
;
            CLRF    INTCON
            CLRF    PIR1
            BSF     STATUS, RP0     ; Bank1
            CLRF    PIE1            ; Disable all peripheral interrupts
;
    if ( C74_REV_A )                ; See PIC16C74 Errata
            BSF     TRISC, T1OSO    ; RC0 needs to be input for the oscillator to function
    endif
            BSF     PIE1, TMR1IE    ; Enable TMR1 Interrupt
;
; Initialize the Special Function Registers (SFR) interrupts
;
            BCF     STATUS, RP0     ; Bank0
            CLRF    PIR1            ;
            BSF     INTCON, PEIE    ; Enable Peripheral Interrupts
            BSF     INTCON, GIE     ; Enable all Interrupts
;
            MOVLW   0x0E
            MOVWF   T1CON           ; Enable T1 Oscillator, Ext Clock, Async, prescaler = 1
            BSF     T1CON, TMR1ON   ; Turn Timer1 ON
;
zzz         SLEEP
            GOTO    zzz             ; Sleep, wait for TMR1 interrupt
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Low-Power Real-Time Clock

Author: *Mark Palmer*
Microchip Technology Inc.

INTRODUCTION

This application note uses the Timer1 module, from a mid-range PIC16CXXX microcontroller, to control a low-power real-time clock. Timer1 was chosen because it has its own crystal which allows the module to operate during sleep. The two events that will wake the device from sleep (for this application) are a keypress and a Timer1 overflow.

OPERATION

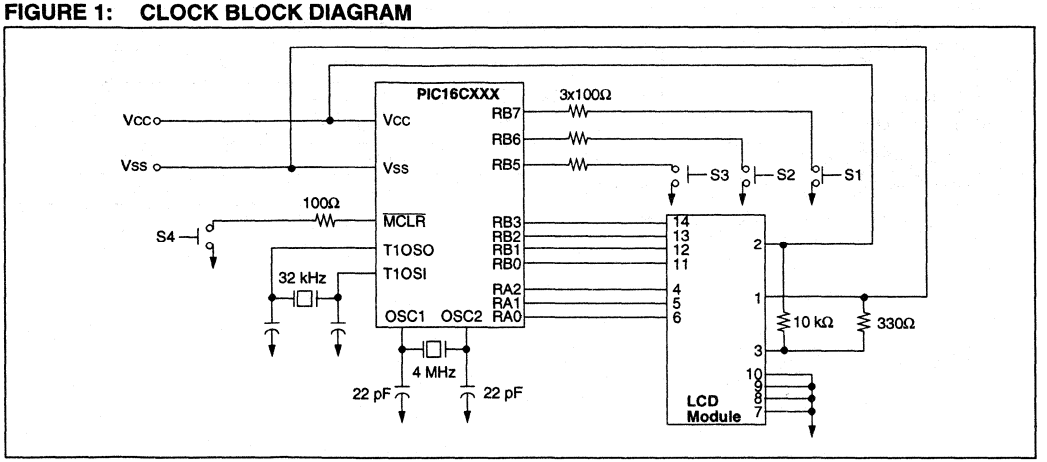
Upon power-up, the device is initialized with the display starting at 12:00 PM, and Timer1 is configured to generate an interrupt (every second). The Timer1 overflow interrupt wakes the device from sleep. This causes the time registers (HRS, MIN, SECS) to be updated. If the SECS register contains an even value (SECS<0> = 0), the colon (":") is not displayed. This gives a visual indication for each second. Then the device returns to sleep.

There are three keys for the setting of the clock. The SELECT_UNITS Key (S1) selects which units are to be modified (hours, minutes, off). The selected units are blanked for a second then flashed for one second. The INC Key (S2) increments the selected units. While incrementing, the selected units values are displayed. Upon key release, the Timer counts out one second and begins flashing the selected units. The CLR_MIN Key (S3) clears the minutes and seconds. CLR_MIN is useful for exactly setting the time to the "top of the hour" as announced in radio broadcasts. After the INC or SELECT_UNITS keys are depressed, the user has ten seconds to depress the next key. If no keypress is detected within ten seconds, the unit returns to the clock mode.

To simplify the design time and minimize cost, a standard Hitachi LCD display module is used. Most applications that require LCDs use a custom LCD display. The LCD interface software would need to be modified to suit the specific LCD display driver being used.

Figure 1 is a block diagram of the design. The RA2:RA0 pins are the control signals to the LCD display, RB3:RB0 acts as a 4-bit data bus, and RB7:RB5 are the input switches. The OSC1 pin is connected to an RC network, which generates an approximate 4 MHz device frequency. Because Timer1 operates asynchronously to the device, the device's oscillator can be configured for RC mode. RC oscillator mode is the least expensive and has the quickest start-up time. Timer1 is where an accurate frequency is required. Timer1's crystal is connected to the T1OSI and T1OSO pins. A good choice for a crystal is a 32.786 kHz (watch) crystal. Table 1 is a list of the components and their part numbers.

FIGURE 1: CLOCK BLOCK DIAGRAM



© 1997 Microchip Technology Inc.

Relative to most microelectronics, LCD's are slow devices. A good portion of the time spent in the Interrupt Service Routine, is talking to and updating the LCD module. To minimize power consumption, the device should be in SLEEP mode as much as possible.

By using the conditional assembly, if a flag (called Debug) is true, the total time spent in the subroutine can be seen on the PORTD<0> pin (the high time). Measuring this time on an oscilloscope displayed a typical time of 800 µs that the device is awake. This 800 µs operation is out of the 1 second time that the device needs to service the interrupt (a Timer1 overflow).

The accuracy of a real-time clock using Timer1 depends on the accuracy of the crystal being used. The more accurate the crystal, the higher the cost. So as always there is a cost / performance trade-off to be made. A crystal rated with an accuracy of 20 PPM (parts per million), could cause an error of about 1.7 seconds per day. For many applications, this should be adequate (said from someone who doesn't wear a watch).

The program written for this application note shows one method for a real-time clock. Trade-offs between code size, current consumption and desired operation have been made. Some possible alternative implementations are:

1. When displaying the time, update only the characters that changed.
2. Turn off the display during sleep
3. LCD module data interface of 8-bits, as-opposed-to the 4-bit interface.

Alternative 1 can reduce the time awake by keeping track of which characters need to be updated. The majority of the time it will be only the position which contains either the ":" or the " ". Next would be the ones place of the minutes, then the tens place of the minutes, etc. The display would only need to be completely updated 2 times every 24 hours. This would reduce the amount of time talking with the LCD display at the cost of some program / data memory.

Depending on the requirements of the application and the characteristics of the display, Alternative 2 could be implemented by turning the power off and on (at a given rate) to the display. This technique may lead to a lower system current consumption. Evaluation of the desired display / display driver is recommended.

Alternative 3 uses the LCD module in an 8-bit mode, which will reduce the size of the display routines (save about 20 words of program memory) at the cost of four additional I/O lines. For some applications this may be a good trade-off to get the additional program memory space. The percentage of operating time saved is slight and should not give substantial power savings.

TABLE 1: LIST OF COMPONENTS†

Description	Part Number	Manufacturer	Quantity
LCD Module (2 x 20 Characters)	LM032L	Hitachi	1
Switches	EVQPADO4M	Panasonic	4
Microcontroller	PIC16C64 / 74	Microchip	1
32.768 kHz Crystal	NC26 / NC38	FOX	1
4 MHz Crystal	ECS-40-20-1	ECS	1

† Most components available from DigiKey.

© 1997 Microchip Technology Inc.

CONCLUSION

The Timer1 module allows many applications to include a real-time clock at minimal system cost. This time function can be useful in consumer applications (display time) as well as in industrial applications (data time stamp). The accuracy of the time is strictly dependent on the accuracy of the crystal. Table 2 shows the program resource requirements.

TABLE 2: PROGRAM RESOURCE REQUIREMENTS

Resource			Words / Bytes	Cycles
Program Memory	Initialization		61	61
	Clock Operation	Increment Time WC	106	35 + Display
		Key Input WC		35 + Display Time
Data Memory	Display[2]		208	526[1]
	Variables		5	N.A.
	Scratch RAM		4	N.A.

[1] Dependent on LCD Module (re; `BUSY_CHECK` subroutine).
[2] Assumes worst case (WC) numbers and best case response from LCD module.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: SOURCE CODE LISTING (CLOCK_01.LST)

```
MPASM 01.40 Released            CLOCK.ASM   1-16-1997  17:05:59         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C74, n = 66
                     00002         ERRORLEVEL  -302
                     00003 ;
                     00004 ;*****************************************************************************
                     00005 ;
                     00006 ; This program implements a real time clock using the TMR1 module of the
                     00007 ; PIC16CXXX family. A LCD display module is used to display (update) the time
                     00008 ; every second. Three keys are used to set the time.
                     00009 ;
                     00010 ;       Program = CLOCK.ASM
                     00011 ;       Revision Date:   5-15-94
                     00012 ;                        1-15-97    Compatibility with MPASMWIN 1.40
                     00013 ;
                     00014 ;*****************************************************************************
                     00015 ;
                     00016 ;
                     00017 ; HARDWARE SETUP
                     00018 ;   LCD Control Lines
                     00019 ;       RA0 = E     (Enable)
                     00020 ;       RA1 = RW    (Read/Write)
                     00021 ;       RA2 = RS    (Register Select)
                     00022 ;   LCD Data Lines
                     00023 ;       RB<3:0>
                     00024 ;   Switch Inputs
                     00025 ;       RB7 = Select Hour / Minute / Off
                     00026 ;       RB6 = Increment Hour / Minute
                     00027 ;       RB5 = Reset Minutes to 00
                     00028 ;
                     00029             INCLUDE <p16c74.inc>
                     00001         LIST
                     00002 ; P16C74.INC  Standard Header File, Version 1.00    Microchip Technology, Inc.
                     00318         LIST
                     00030
  00000000           00031 FALSE                   EQU             0
  00000001           00032 TRUE                    EQU             1
```


© 1997 Microchip Technology Inc.

```
                        00033
                        00034             INCLUDE <CLOCK.h>
                        00076         list
                        00035 ;
  00000006              00036 LCD_DATA        EQU     PORTB           ; The LCD data is on the lower 4-bits
  00000086              00037 LCD_DATA_TRIS   EQU     TRISB           ; The TRIS register for the LCD data
  00000005              00038 LCD_CNTL        EQU     PORTA           ; Three control lines
                        00039 ;
  00000000              00040 PICMaster       EQU     FALSE           ; A Debugging Flag
  00000000              00041 Debug           EQU     FALSE           ; A Debugging Flag
  00000001              00042 Debug_PU        EQU     TRUE            ; A Debugging Flag
                        00043 ;
                        00044 ;
                        00045 ; Reset address. Determine type of RESET
                        00046 ;
0000                    00047         org     RESET_V                 ; RESET vector location
0000 1683               00048 RESET       BSF     STATUS, RP0         ; Bank 1
0001 188E               00049             BTFSC   PCON, NOT_POR       ; Power-up reset?
0002 290C               00050             GOTO    START               ; YES
0003 295E               00051             GOTO    OTHER_RESET         ; NO, a WDT or MCLR reset
                        00052 ;
                        00053 ; This is the Periperal Interrupt routine. Need to determine the type
                        00054 ; of interrupt that occurred. The following interrupts are enabled:
                        00055 ;   1.  PORTB Change (RBIF)
                        00056 ;   2.  TMR1 Overflow Interrupt (T1IF)
                        00058     page
0004                    00059         org     ISR_V                   ; Interrupt vector location
0004                    00060 PER_INT_V
                        00061         if ( Debug )
                        00062             bsf     PORTD, 0            ; Set high, use to measure total
                        00063         endif                           ;     time in Int Service Routine
                        00064 ;
0004 1283               00065             BCF         STATUS, RP0     ; Bank 0
0005 180C               00066             BTFSC       PIR1, TMR1IF    ; Timer 1 overflowed?
0006 2843               00067             GOTO        T1_OVRFL        ; YES, Service the Timer1 Overflow Interrupt
0007 1C0B               00068             BTFSS       INTCON, RBIF    ; NO, Did PORTB change?
0008 28D0               00069             GOTO        ERROR1          ; NO, Error Condition - Unknown Interrupt
                        00070 ;
0009                    00071 PORTB_FLAG                              ; Are any of PORTB's inputs active?
0009 0806               00072             MOVF        PORTB, W        ;
000A 39E0               00073             ANDLW       0xE0            ; Keep only the 3 switch values
000B 00B5               00074 DEBOUNCE    MOVWF       TEMP            ;
000C 3002               00075             MOVLW       DB_HI_BYTE      ; This is the debounce delay
000D 08B3               00076             MOVF        MSD, F          ;
000E 01B4               00077             CLRF        LSD             ;
000F 0BB4               00078 KB_D_LP1    DECFSZ      LSD, F          ;
0010 280F               00079             GOTO        KB_D_LP1        ;
```

© 1997 Microchip Technology Inc.

```
0011 0BB3           00080             DECFSZ      MSD, F          ;
0012 280F           00081             GOTO        KB_D_LP1        ;
0013 0806           00082 END_DELAY   MOVF        PORTB, W        ;
0014 39E0           00083             ANDLW       0xE0            ; Keep only the 3 switch values
0015 02B5           00084             SUBWF       TEMP, F         ;
0016 1D03           00085             BTFSS       STATUS, Z       ; Is the Zero bit set?
                    00086                                         ;    (switches were the same on 2 reads)
0017 280B           00087             GOTO        DEBOUNCE        ; NO, Try another read
0018 00B5           00088 KEY_MATCH   MOVWF       TEMP            ; YES, need to see which is depressed.
                    00089 ;
0019 3080           00090             MOVLW       0x80            ; Since doing key inputs, clear TMR1
001A 008F           00091             MOVWF       TMR1H           ;   for 1 sec overflow.
001B 018E           00092             CLRF        TMR1L           ;
001C 100C           00093             BCF         PIR1, TMR1IF    ; Clear Timer 1 Interrupt Flag
                    00094
001D 1FB5           00095             BTFSS       TEMP, HR_MIN_SW         ; Is the hour-min-off switch depressed?
001E 2826           00096             GOTO        SELECT_UNITS            ; YES, specify the units selected
001F 1F35           00097             BTFSS       TEMP, INC_SW            ; Is the inc switch depressed?
0020 282B           00098             GOTO        INC_UNIT                ; YES, Increment the selected Units
0021 1EB5           00099             BTFSS       TEMP, CLR_MIN_SW        ; Is the clear minute switch depressed?
0022 2835           00100             GOTO        CLR_MIN                 ; YES, clear the minutes.
                    00101 ;
                    00102 ; No key match occured, or finished with PortB interrupt and need to clear interrupt condition.
                    00103 ;
0023                00104 CLR_RB                                        ; No RB<7:5> keys are depressed (rising edge Int.)
0023 0886           00105             MOVF        PORTB, F        ; Clear the PORTB mismatch condition
0024 100B           00106             BCF         INTCON, RBIF    ; Clear the PORTB Int Flag
                    00107         if ( Debug )
                    00108             bcf         PORTD, 0        ; Set low, use to measure total
                    00109                                         ;     time in Int Service Routine
                    00110         endif
0025 0009           00111             RETFIE                      ; Return / Enable Global Interrupts
                    00112 ;
                    00113     page
0026                00114 SELECT_UNITS
0026 30FF           00115             MOVLW       0xFF            ;
0027 00C0           00116             MOVWF       WAIT_CNTR       ; WAIT_CNTR has LSb set after each SELECT UNIT key press.
0028 0AA0           00117             INCF        FLAG_REG, F             ; Increment the pointer to the MIN_UNIT:HR_UNIT
0029 1620           00118             BSF         FLAG_REG, KEY_INPUT     ;
002A 2875           00119             GOTO        DISPLAY                 ; Flash the Display of the selected unit
                    00120 ;
002B                00121 INC_UNIT
002B 01C0           00122             CLRF        WAIT_CNTR               ; WAIT_CNTR is cleared to zero after each key press.
002C 1820           00123             BTFSC       FLAG_REG, HR_UNIT       ; Are the hour units selected?
002D 285C           00124             GOTO        INC_HRS                 ; YES, Increment the hour units
002E 1CA0           00125             BTFSS       FLAG_REG, MIN_UNIT      ; Are the minute units selected?
002F 2823           00126             GOTO        CLR_RB                  ; NO, Not a valid key. Clear flags
```


© 1997 Microchip Technology Inc.

```
                        00127 ;
0030 0AB1               00128           INCF        MIN, F                  ; YES, Increment the minute units
0031 303C               00129           MOVLW       0x3C                    ; This is Decimal 60
0032 0231               00130           SUBWF       MIN, W                  ; MIN - 60 = ?
0033 1D03               00131           BTFSS       STATUS, Z               ; MIN = 60?
0034 2875               00132           GOTO        DISPLAY                 ; NO, display time
                        00133                                               ; YES, MIN = 0 (use code from CLR_MIN)
0035 01B1               00134 CLR_MIN   CLRF        MIN                     ; MIN = 0
0036 3004               00135           MOVLW       0x04                    ; Clear the seconds
0037 00B2               00136           MOVWF       SECS                    ; Initial Second count = 4
0038 3080               00137           MOVLW       0x80                    ; Clear Timer 1, for 1 sec overflow
0039 008F               00138           MOVWF       TMR1H                   ;
003A 018E               00139           CLRF        TMR1L                   ;
003B 100C               00140           BCF         PIR1, TMR1IF            ; Clear the TMR1 overflow interrupt.
003C 01C0               00141           CLRF        WAIT_CNTR               ; WAIT_CNTR is cleared to zero after each key press.
003D 1AB5               00142           BTFSC       TEMP, CLR_MIN_SW        ; Is the clear minute switch depressed?
003E 2875               00143           GOTO        DISPLAY                 ; NO. Rollover from increment key
003F 10A0               00144           BCF         FLAG_REG, MIN_UNIT      ; YES, Clear ALL relevant flags
0040 1020               00145           BCF         FLAG_REG, HR_UNIT       ;
0041 1220               00146           BCF         FLAG_REG, KEY_INPUT     ;
0042 2875               00147           GOTO        DISPLAY                 ;
                        00148 ;
                        00149     page
                        00150 ;
0043                    00151 T1_OVRFL
0043 100C               00152           BCF         PIR1, TMR1IF            ; Clear Timer 1 Interrupt Flag
0044 1E20               00153           BTFSS       FLAG_REG, KEY_INPUT     ; Are we using the key inputs?
0045 284F               00154           GOTO        INC_TIME                ; NO, Need to Increment the time
0046 0AC0               00155           INCF        WAIT_CNTR, F            ; YES,
0047 300A               00156           MOVLW       0x0A                    ; 10 counts x 1 seconds
0048 0240               00157           SUBWF       WAIT_CNTR, W            ; Has the 10 Sec wait for key expired?
0049 1D03               00158           BTFSS       STATUS, Z               ; Is the result 0?
004A 2875               00159           GOTO        DISPLAY                 ; NO, Display value
004B 01C0               00160           CLRF        WAIT_CNTR               ; YES, Clear WAIT_CNTR
004C 1220               00161           BCF         FLAG_REG, KEY_INPUT     ;
004D 1020               00162           BCF         FLAG_REG, HR_UNIT       ;
004E 10A0               00163           BCF         FLAG_REG, MIN_UNIT      ;
                        00164 ;
                        00165 ;
004F 3080               00166 INC_TIME      MOVLW   0x80                    ;
0050 008F               00167               MOVWF   TMR1H           ; 1 Second Overflow
0051 0AB2               00168               INCF    SECS, F         ;
0052 1F32               00169               BTFSS   SECS, 6         ;
0053 2875               00170               GOTO    DISPLAY         ;
0054 3004               00171               MOVLW   0x04            ;
0055 00B2               00172               MOVWF   SECS            ;
0056 0AB1               00173               INCF    MIN, F          ;
```

© 1997 Microchip Technology Inc.

```
0057 303C          00174                MOVLW   0x3C            ; W = 60d
0058 0231          00175                SUBWF   MIN, W          ;
0059 1D03          00176                BTFSS   STATUS, Z       ;
005A 2875          00177                GOTO    DISPLAY         ;
005B 01B1          00178                CLRF    MIN             ;
005C 0AB0          00179 INC_HRS        INCF    HRS, F          ;
                   00180
005D 300C          00181                MOVLW   0x0C            ; It is now 12:00, Toggle AM / PM
005E 0230          00182                SUBWF   HRS, W          ;
005F 1D03          00183                BTFSS   STATUS, Z       ;
0060 2867          00184                GOTO    CK_13           ; Need to check if HRS = 13
0061 1FA0          00185                BTFSS   FLAG_REG, AM    ; Was it AM or PM
0062 2865          00186                GOTO    SET_AM          ; Was PM, Needs to be AM
0063 13A0          00187                BCF     FLAG_REG, AM    ; It is PM
0064 2875          00188                GOTO    DISPLAY         ;
0065 17A0          00189 SET_AM         BSF     FLAG_REG, AM    ; It is AM
0066 2875          00190                GOTO    DISPLAY         ;
                   00191
0067 300D          00192 CK_13          MOVLW   0x0D                    ; Check if HRS = 13
0068 0230          00193                SUBWF   HRS, W                  ;
0069 1D03          00194                BTFSS   STATUS, Z               ;
006A 2875          00195                GOTO    DISPLAY                 ;
006B 01B0          00196                CLRF    HRS                     ;
006C 0AB0          00197                INCF    HRS, F                  ;
006D 2875          00198                GOTO    DISPLAY                 ;
                   00199 ;
                   00200        page
006E               00201 INIT_DISPLAY
006E 300C          00202                MOVLW   DISP_ON                 ; Display On, Cursor On
006F 20E3          00203                CALL    SEND_CMD                ; Send This command to the Display Module
0070 3001          00204                MOVLW   CLR_DISP                ; Clear the Display
0071 20E3          00205                CALL    SEND_CMD                ; Send This command to the Display Module
0072 3006          00206                MOVLW   ENTRY_INC               ; Set Entry Mode Inc., No shift
0073 20E3          00207                CALL    SEND_CMD                ; Send This command to the Display Module
0074 0008          00208                RETURN
                   00209 ;
0075               00210 DISPLAY
0075 3080          00211                MOVLW   DD_RAM_ADDR             ;
0076 20E3          00212                CALL    SEND_CMD                ;
                   00213 ;
0077 1A20          00214                BTFSC   FLAG_REG, KEY_INPUT     ; Do we need to flash the selected units?
0078 287D          00215                GOTO    FLASH_UNITS             ; YES, we need to flash selected units
0079 20A4          00216                CALL    LOAD_HRS                ; NO, do a normal display
007A 20AD          00217                CALL    LOAD_COLON              ;
007B 20B2          00218                CALL    LOAD_MIN                ;
007C 28BB          00219                GOTO    LOAD_AM                 ;
                   00220 ;
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

```
007D                00221 FLASH_UNITS
007D 018A           00222                 CLRF    PCLATH                  ; This clears PCLATH, This table in 1st
007E 0820           00223                 MOVF    FLAG_REG, W             ;    256 bytes of program memory
007F 3903           00224                 ANDLW   0x03                    ; only HR_UNIT and MIN_UNIT bit can be non-zero
0080                00225 UNIT_TBL
0080 0782           00226                 ADDWF   PCL, F                  ; HR_UNIT:MIN_UNIT
0081 289F           00227                 GOTO    NO_UNITS                ;       0 0        - Display everything.
0082 2887           00228                 GOTO    HR_UNITS                ;       0 1        - Flash the hour units
0083 2893           00229                 GOTO    MIN_UNITS               ;       1 0        - Flash the minute units
0084                00230 UNIT_TBL_END
0084 30FC           00231                 MOVLW   0xFC                    ;       1 1        - Need to clear FLAG_REG<HR_UNIT:MIN_UN
                         IT>
0085 05A0           00232                 ANDWF   FLAG_REG, F             ;
0086 289F           00233                 GOTO    NO_UNITS                ;       0 0        - Display everything.
                    00234 ;
                    00235         if ( (UNIT_TBL & 0x0FF) >= (UNIT_TBL_END & 0x0FF) )
                    00236        MESSG   "Warning: Table UNIT_TBL crosses page boundry in computed jump"
                    00237     endif
                    00238 ;
                    00239 ;
0087                00240 HR_UNITS
0087 1C40           00241                 BTFSS   WAIT_CNTR, 0            ; If WAIT_CNTR is odd,
                    00242                                                 ;    hour digits are displayed as blank
0088 288D           00243                 GOTO    SKIP_BLK_HRS            ;
0089 3020           00244                 MOVLW   ' '                     ;
008A 20D4           00245                 CALL    SEND_CHAR               ;
008B 3020           00246                 MOVLW   ' '                     ;
008C 20D4           00247                 CALL    SEND_CHAR               ;
008D                00248 SKIP_BLK_HRS
008D 1C40           00249                 BTFSS   WAIT_CNTR, 0            ; WAIT_CNTR was even, display hour digits
008E 20A4           00250                 CALL    LOAD_HRS                ;
                    00251 ;
008F 303A           00252                 MOVLW   ':'                     ; : always on, display all other character
0090 20D4           00253                 CALL    SEND_CHAR               ;
0091 20B2           00254                 CALL    LOAD_MIN                ;
0092 28BB           00255                 GOTO    LOAD_AM                 ;
                    00256 ;
                    00257         page
0093                00258 MIN_UNITS
0093 20A4           00259                 CALL    LOAD_HRS                ; Display hours
0094 303A           00260                 MOVLW   ':'                     ; : always on
0095 20D4           00261                 CALL    SEND_CHAR               ;
0096 1C40           00262                 BTFSS   WAIT_CNTR, 0            ; If WAIT_CNTR is odd,
                    00263                                                 ;    minute digits are displayed as blank
0097 289C           00264                 GOTO    SKIP_BLK_MIN            ;
0098 3020           00265                 MOVLW   ' '                     ;
0099 20D4           00266                 CALL    SEND_CHAR               ;
```

```
009A 3020          00267                MOVLW   ' '                ;
009B 20D4          00268                CALL    SEND_CHAR          ;
009C               00269 SKIP_BLK_MIN
009C 1C40          00270                BTFSS   WAIT_CNTR, 0       ; WAIT_CNTR was even, display minute digits
009D 20B2          00271                CALL    LOAD_MIN           ;
009E 28BB          00272                GOTO    LOAD_AM            ;
                   00273 ;
009F               00274 NO_UNITS
009F 20A4          00275                CALL    LOAD_HRS           ; Display all character
00A0 303A          00276                MOVLW   ':'                ;
00A1 20D4          00277                CALL    SEND_CHAR          ;
00A2 20B2          00278                CALL    LOAD_MIN           ;
00A3 28BB          00279                GOTO    LOAD_AM            ;
                   00280 ;
00A4               00281 LOAD_HRS
00A4 0830          00282                MOVF    HRS, W             ; Load the Wreg with the value
00A5 20C7          00283                CALL    BIN_2_BCD          ;   to convert to BCD
00A6 0833          00284                MOVF    MSD, W             ; Load the MSD value into the Wreg
00A7 2400          00285                CALL    NUM_TABLE          ; Get the ASCII code
00A8 20D4          00286                CALL    SEND_CHAR          ; Send this Character to the Display
                   00287 ;
00A9 0834          00288                MOVF    LSD, W             ; Load the LSD value into the Wreg
00AA 2400          00289                CALL    NUM_TABLE          ; Get the ASCII code
00AB 20D4          00290                CALL    SEND_CHAR          ; Send this Character to the Display
00AC 0008          00291                RETURN
                   00292 ;
00AD 3020          00293 LOAD_COLON     MOVLW   ' '                ; ASCII value for a Blank space
00AE 1832          00294                BTFSC   SECS, 0            ; Is it an EVEN or ODD second
00AF 3E1A          00295                ADDLW   ':' - ' '          ; Is ODD, Second colon is ON.
                   00296                                           ;   Add delta offset of ASCII Characters
00B0 20D4          00297                CALL    SEND_CHAR          ; Send this Character to the Display
00B1 0008          00298                RETURN
                   00299 ;
00B2               00300 LOAD_MIN
00B2 0831          00301                MOVF    MIN, W             ; Load the Wreg with the value
00B3 20C7          00302                CALL    BIN_2_BCD          ;   to convert to BCD
00B4 0833          00303                MOVF    MSD, W             ; Load the MSD value into the Wreg
00B5 2400          00304                CALL    NUM_TABLE          ; Get the ASCII code
00B6 20D4          00305                CALL    SEND_CHAR          ; Send this Character to the Display
                   00306 ;
00B7 0834          00307                MOVF    LSD, W             ; Load the LSD value into the Wreg
00B8 2400          00308                CALL    NUM_TABLE          ; Get the ASCII code
00B9 20D4          00309                CALL    SEND_CHAR          ; Send this Character to the Display
00BA 0008          00310                RETURN
                   00311 ;
                   00312        page
00BB 3020          00313 LOAD_AM        MOVLW   ' '                ; ASCII value for a Blank space
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
00BC 20D4          00314                CALL    SEND_CHAR               ; Send this Character to the Display
00BD 3041          00315                MOVLW   'A'                     ; ASCII value for a Blank space
00BE 1FA0          00316                BTFSS   FLAG_REG, AM            ; Is it AM or PM
00BF 3E0F          00317                ADDLW   'P' - 'A'               ; Is PM, Add delta offset of ASCII Characters
00C0 20D4          00318                CALL    SEND_CHAR               ; Send this Character to the Display
00C1 304D          00319                MOVLW   'M'
00C2 20D4          00320                CALL    SEND_CHAR               ; Send this Character to the Display
                   00321 ;
00C3 1683          00322                BSF     STATUS, RP0             ; Bank 1
00C4 1381          00323                BCF     OPTION_REG, NOT_RBPU            ; Turn on PORTB Pull-up
00C5 1283          00324                BCF     STATUS, RP0             ; Bank 0
00C6 2823          00325                GOTO    CLR_RB                  ; You've displayed the time, Clear RBIF
                   00326 ;
                   00327 ;
                   00328 ;*****************************************************************************
                   00329 ; The BIN_2_BCD routine converts the binary number, in the W register, to a
                   00330 ; binary coded decimal (BCD) munber. This BCD number is stored MSD:LSD. This
                   00331 ; routine is used by the DISPLAY subroutine, to convert the time values.
                   00332 ;*****************************************************************************
                   00333 ;
00C7 01B3          00334 BIN_2_BCD      CLRF    MSD                     ; This value contain the 10's digit value
00C8 00B4          00335                MOVWF   LSD                     ; This value contain the 1's digit value
00C9 300A          00336 TENS_SUB       MOVLW   .10                     ; A decimal 10
00CA 0234          00337                SUBWF   LSD, W                  ;
00CB 1C03          00338                BTFSS   STATUS, C               ; Did this subtract cause a Negative Result?
00CC 3400          00339                RETLW   0                       ; YES, Return from this Routine
00CD 00B4          00340                MOVWF   LSD                     ; No, move the result into LSD
00CE 0AB3          00341                INCF    MSD, F                  ; Increment the most significat digit
00CF 28C9          00342                GOTO    TENS_SUB                ;
                   00343 ;
                   00344 ;
                   00345 ; Should NEVER get here
                   00346 ;
00D0 1283          00347 ERROR1          BCF     STATUS, RP0         ; Bank 0
                   00348 ;
                   00349           if ( Debug )
                   00350               BSF     PORTD, 1
                   00351               BCF     PORTD, 1
                   00352           else
00D1 1407          00353               BSF     PORTC, 0
00D2 1007          00354               BCF     PORTC, 0
                   00355           endif
00D3 28D0          00356               GOTO    ERROR1
                   00357 ;
                   00358           page
                   00359 ;
                   00360 ;*****************************************************************
```

```
                        00361 ;* SendChar - Sends character to LCD                                  *
                        00362 ;* This routine splits the character into the upper and lower        *
                        00363 ;* nibbles and sends them to the LCD, upper nibble first.           *
                        00364 ;* The data is transmitted on the PORT<3:0> pins                     *
                        00365 ;*************************************************************************
                        00366
00D4                    00367 SEND_CHAR
00D4 00B6               00368             MOVWF   CHAR                ; Character to be sent is in W
00D5 20F2               00369             CALL    BUSY_CHECK          ; Wait for LCD to be ready
00D6 0E36               00370             SWAPF   CHAR, W
00D7 390F               00371             ANDLW   0x0F                ; Get upper nibble
00D8 0086               00372             MOVWF   LCD_DATA            ; Send data to LCD
00D9 1085               00373             BCF     LCD_CNTL, RW        ; Set LCD to read
00DA 1505               00374             BSF     LCD_CNTL, RS        ; Set LCD to data mode
00DB 1405               00375             BSF     LCD_CNTL, E         ; toggle E for LCD
00DC 1005               00376             BCF     LCD_CNTL, E
00DD 0836               00377             MOVF    CHAR, W
00DE 390F               00378             ANDLW   0x0F                ; Get lower nibble
00DF 0086               00379             MOVWF   LCD_DATA            ; Send data to LCD
00E0 1405               00380             BSF     LCD_CNTL, E         ; toggle E for LCD
00E1 1005               00381             BCF     LCD_CNTL, E
00E2 0008               00382             RETURN
                        00383
                        00384 ;*************************************************************************
                        00385 ;* SendCmd - Sends command to LCD                                     *
                        00386 ;* This routine splits the command into the upper and lower           *
                        00387 ;* nibbles and sends them to the LCD, upper nibble first.           *
                        00388 ;* The data is transmitted on the PORT<3:0> pins                     *
                        00389 ;*************************************************************************
                        00390
00E3                    00391 SEND_CMD
00E3 00B6               00392             MOVWF   CHAR                ; Character to be sent is in W
00E4 20F2               00393             CALL    BUSY_CHECK          ; Wait for LCD to be ready
00E5 0E36               00394             SWAPF   CHAR, W
00E6 390F               00395             ANDLW   0x0F                ; Get upper nibble
00E7 0086               00396             MOVWF   LCD_DATA            ; Send data to LCD
00E8 1085               00397             BCF     LCD_CNTL, RW        ; Set LCD to read
00E9 1105               00398             BCF     LCD_CNTL, RS        ; Set LCD to command mode
00EA 1405               00399             BSF     LCD_CNTL, E         ; toggle E for LCD
00EB 1005               00400             BCF     LCD_CNTL, E
00EC 0836               00401             MOVF    CHAR, W
00ED 390F               00402             ANDLW   0x0F                ; Get lower nibble
00EE 0086               00403             MOVWF   LCD_DATA            ; Send data to LCD
00EF 1405               00404             BSF     LCD_CNTL, E         ; toggle E for LCD
00F0 1005               00405             BCF     LCD_CNTL, E
00F1 0008               00406             RETURN
                        00407     page
```


© 1997 Microchip Technology Inc.

```
                        00408 ;*******************************************************************
                        00409 ;* This routine checks the busy flag, returns when not busy        *
                        00410 ;*  Affects:                                                        *
                        00411 ;*      TEMP - Returned with busy/address                          *
                        00412 ;*******************************************************************
                        00413
00F2                    00414 BUSY_CHECK
                        00415 ;
                        00416         if ( Debug )
                        00417             BSF     PORTD, 3
                        00418             BCF     PORTD, 3
                        00419         endif
00F2 0186               00420             CLRF    LCD_DATA            ;** Have PORTB<3:0> output low
00F3 1683               00421             BSF     STATUS, RP0         ; Bank 1
00F4 1781               00422             BSF     OPTION_REG, NOT_RBPU      ; Turn off PORTB Pull-up
00F5 30FF               00423             MOVLW   0xFF                ; Set PortB for input
00F6 0086               00424             MOVWF   LCD_DATA_TRIS
00F7 1283               00425             BCF     STATUS, RP0         ; Bank 0
00F8 1105               00426             BCF     LCD_CNTL, RS        ; Set LCD for Command mode
00F9 1485               00427             BSF     LCD_CNTL, RW        ; Setup to read busy flag
00FA 1405               00428             BSF     LCD_CNTL, E         ; Set E high
00FB 1005               00429             BCF     LCD_CNTL, E         ; Set E low
00FC 0E06               00430             SWAPF   LCD_DATA, W         ; Read upper nibble busy flag, DDRam address
00FD 39F0               00431             ANDLW   0xF0                ; Mask out lower nibble
00FE 00B5               00432             MOVWF   TEMP                ;
00FF 1405               00433             BSF     LCD_CNTL, E         ; Toggle E to get lower nibble
0100 1005               00434             BCF     LCD_CNTL, E
0101 0806               00435             MOVF    LCD_DATA, W         ; Read lower nibble busy flag, DDRam address
0102 390F               00436             ANDLW   0x0F                ; Mask out upper nibble
0103 04B5               00437             IORWF   TEMP, F             ; Combine nibbles
0104 1BB5               00438             BTFSC   TEMP, 7             ; Check busy flag, high = busy
0105 28F2               00439             GOTO    BUSY_CHECK          ; If busy, check again
0106 1085               00440             BCF     LCD_CNTL, RW
0107 1683               00441             BSF     STATUS, RP0         ; Bank 1
0108 30F0               00442             MOVLW   0xF0                ;
0109 0086               00443             MOVWF   LCD_DATA_TRIS       ; RB7 - 4 = inputs, RB3 - 0 = output
010A 1283               00444             BCF     STATUS, RP0         ; Bank 0
010B 0008               00445             RETURN
                        00446 ;
                        00447     page
                        00448 ;
                        00449 ;*******************************************************************************
                        00450 ;*****       Start program here, Power-On Reset occurred.
                        00451 ;*******************************************************************************
                        00452 ;
010C                    00453 START                               ; POWER_ON Reset (Beginning of program)
010C 1283               00454             BCF     STATUS, RP0     ; Bank 0
```

© 1997 Microchip Technology Inc.

```
010D 300C           00455              MOVLW   0x0C            ; Decimal 12
010E 00B0           00456              MOVWF   HRS             ; HOURS = 12
010F 01B1           00457              CLRF    MIN             ; MIN   = 00
0110 3000           00458              MOVLW   0x00            ;
0111 00A0           00459              MOVWF   FLAG_REG        ; PM light is on
0112 3004           00460              MOVLW   0x04            ; Initial value of seconds (64d - 60d)
0113 00B2           00461              MOVWF   SECS            ; This allows a simple bit test to see if 60
                    00462                                      ;   secs has elapsed.
0114 3080           00463              MOVLW   0x80            ; TIM1H:TMR1L = 0x8000 gives 1 second
0115 008F           00464              MOVWF   TMR1H           ;   overflow, at 32 KHz.
0116 018E           00465              CLRF    TMR1L           ;
                    00466 ;
0117                00467 MCLR_RESET                           ; A Master Clear Reset
0117 0183           00468              CLRF    STATUS          ; Do initialization (Bank 0)
0118 018B           00469              CLRF    INTCON
0119 018C           00470              CLRF    PIR1
011A 1683           00471              BSF     STATUS, RP0     ; Bank 1
011B 3000           00472              MOVLW   0x00            ; The LCD module does not like to work w/ weak pull-ups
011C 0081           00473              MOVWF   OPTION_REG      ;
011D 018C           00474              CLRF    PIE1            ; Disable all peripheral interrupts
011E 30FF           00475              MOVLW   0xFF            ;
011F 009F           00476              MOVWF   ADCON1          ; Port A is Digital (for 16C7x devices).
                    00477 ;
                    00478 ;
0120 1283           00479              BCF     STATUS, RP0     ; Bank 0
0121 0185           00480              CLRF    PORTA           ; ALL PORT output should output Low.
0122 0186           00481              CLRF    PORTB
0123 0187           00482              CLRF    PORTC
0124 0188           00483              CLRF    PORTD
0125 0189           00484              CLRF    PORTE
0126 1010           00485              BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
                    00486 ;
0127 1683           00487              BSF     STATUS, RP0     ; Select Bank 1
0128 0185           00488              CLRF    TRISA           ; RA5 -  0 outputs
0129 30F0           00489              MOVLW   0xF0            ;
012A 0086           00490              MOVWF   TRISB           ; RB7 - 4 inputs, RB3 - 0 outputs
012B 0187           00491              CLRF    TRISC           ; RC Port are outputs
012C 1407           00492              BSF     TRISC, T1OSO    ; RC0 needs to be input for the oscillator to function
012D 0188           00493              CLRF    TRISD           ; RD Port are outputs
012E 0189           00494              CLRF    TRISE           ; RE Port are outputs
012F 140C           00495              BSF     PIE1, TMR1IE    ; Enable TMR1 Interrupt
0130 1381           00496              BCF     OPTION_REG, NOT_RBPU  ; Enable PORTB pull-ups
0131 1283           00497              BCF     STATUS, RP0     ; Select Bank 0
0132 0886           00498              MOVF    PORTB, F        ; Need to clear 1st RBIF, due to
0133 100B           00499              BCF     INTCON, RBIF    ;    set up of PORTB
                    00500 ;
                    00501     page
```


© 1997 Microchip Technology Inc.

```
                    00502 ;
                    00503 ; Initilize the LCD Display Module
                    00504 ;
0134 0185           00505               CLRF    LCD_CNTL        ; ALL PORT output should output Low.
                    00506
0135                00507 DISPLAY_INIT
0135 3002           00508               MOVLW   0x02            ; Command for 4-bit interface
0136 0086           00509               MOVWF   LCD_DATA        ;
0137 1405           00510               BSF     LCD_CNTL, E     ;
0138 1005           00511               BCF     LCD_CNTL, E     ;
                    00512 ;
                    00513 ; This routine takes the calculated times that the delay loop needs to
                    00514 ; be executed, based on the LCD_INIT_DELAY EQUate that includes the
                    00515 ; frequency of operation. It uses registers before they are needed to
                    00516 ; store the time.
                    00517 ;
0139 3006           00518 LCD_DELAY   MOVLW   LCD_INIT_DELAY  ;
013A 00B3           00519               MOVWF   MSD             ; Use MSD and LSD Registers to Initialize LCD
013B 01B4           00520               CLRF    LSD             ;
013C 0BB4           00521 LOOP2       DECFSZ  LSD, F          ; Delay time = MSD * ((3 * 256) + 3) * Tcy
013D 293C           00522               GOTO    LOOP2           ;
013E 0BB3           00523               DECFSZ  MSD, F          ;
013F                00524 END_LCD_DELAY
013F 293C           00525               GOTO    LOOP2           ;
                    00526 ;
                    00527 ; Command sequence for 2 lines of 5x7 characters
                    00528 ;
0140 3002           00529 CMD_SEQ     MOVLW   0X02
0141 0086           00530               MOVWF   LCD_DATA
0142 1405           00531               BSF     LCD_CNTL, E     ;
0143 1005           00532               BCF     LCD_CNTL, E     ;
0144 3008           00533               MOVLW   0x08            ;
0145 0086           00534               MOVWF   LCD_DATA        ;
0146 1405           00535               BSF     LCD_CNTL, E     ;
0147 1005           00536               BCF     LCD_CNTL, E     ;
                    00537 ;
                    00538 ; Busy Flag should be valid after this point
                    00539 ;
0148 300C           00540               MOVLW   DISP_ON         ;
0149 20E3           00541               CALL    SEND_CMD        ;
014A 3001           00542               MOVLW   CLR_DISP        ;
014B 20E3           00543               CALL    SEND_CMD        ;
014C 3006           00544               MOVLW   ENTRY_INC       ;
014D 20E3           00545               CALL    SEND_CMD        ;
014E 3080           00546               MOVLW   DD_RAM_ADDR     ;
014F 20E3           00547               CALL    SEND_CMD        ;
                    00548 ;
```

© 1997 Microchip Technology Inc.

```
                        00549     page
                        00550 ;
                        00551 ; Initialize the Special Function Registers (SFR) interrupts
                        00552 ;
0150 018C               00553             CLRF    PIR1            ;
0151 300E               00554             MOVLW   0x0E
0152 0090               00555             MOVWF   T1CON           ; RC1 is overridden by TCKO
0153 170B               00556             BSF     INTCON, PEIE    ; Enable Peripheral Interrupts
0154 158B               00557             BSF     INTCON, RBIE    ; Disable PORTB<7:4> Change Interrupts
0155 178B               00558             BSF     INTCON, GIE     ; Enable all Interrupts
                        00559 ;
0156 206E               00560             CALL    INIT_DISPLAY    ;
0157 2075               00561             CALL    DISPLAY         ;
                        00562 ;
0158 300E               00563             MOVLW   0x0E
0159 0090               00564             MOVWF   T1CON           ; Enable T1 Oscillator, Ext Clock, Async, prescaler = 1
015A 1410               00565             BSF     T1CON, TMR1ON   ; Turn Timer 1 ON
                        00566 ;
                        00567         if ( PICMaster )
                        00568 lzz         goto    lzz             ; Loop waiting for interrupts (for use with PICMASTER)
                        00569         else
                        00570 ;
015B 0063               00571 SLEEP_LP    SLEEP                   ; Wait for Change on PORTB interrupt. or TMR1 timeout
015C 0000               00572             NOP                     ;
015D 295B               00573             GOTO    SLEEP_LP        ;
                        00574 ;
                        00575         endif
                        00576 ;
                        00577 ; Here is where you do things depending on the type of RESET (Not a Power-On Reset).
                        00578 ;
015E 1E03               00579 OTHER_RESET   BTFSS  STATUS,NOT_TO  ; WDT Time-out?
015F 28D0               00580 WDT_TIMEOUT   GOTO   ERROR1        ; YES, This is error condition
                        00581         if ( Debug_PU )
0160 290C               00582             goto   START           ; MCLR reset, Goto START
                        00583         else
                        00584             GOTO   MCLR_RESET      ; MCLR reset, Goto MCLR_RESET
                        00585         endif
                        00586 ;
                        00587         if (Debug )
                        00588 END_START     NOP                  ; END label for debug
                        00589         endif
                        00590 ;
                        00591     page
                        00592 ;
0400                    00593     org     TABLE_ADDR
                        00594 ;
0400 00B5               00595 NUM_TABLE       MOVWF   TEMP        ; Store value to TEMP register
```


© 1997 Microchip Technology Inc.

```
0401 3004           00596                MOVLW   HIGH (TABLE_ADDR)   ; Ensure that PCLATH high has the
0402 008A           00597                MOVWF   PCLATH              ;   correct value
0403 0835           00598                MOVF    TEMP, W             ; Value into table
0404 390F           00599                ANDLW   0x0F                ; Mask to 4-bits (00 - 0Fh)
0405 0782           00600 NUM_TBL        ADDWF   PCL, F              ; Determine Offset into table
0406 3430           00601                RETLW   '0'                 ; ASCII value of "0" in W register
0407 3431           00602                RETLW   '1'                 ; ASCII value of "1" in W register
0408 3432           00603                RETLW   '2'                 ; ASCII value of "2" in W register
0409 3433           00604                RETLW   '3'                 ; ASCII value of "3" in W register
040A 3434           00605                RETLW   '4'                 ; ASCII value of "4" in W register
040B 3435           00606                RETLW   '5'                 ; ASCII value of "5" in W register
040C 3436           00607                RETLW   '6'                 ; ASCII value of "6" in W register
040D 3437           00608                RETLW   '7'                 ; ASCII value of "7" in W register
040E 3438           00609                RETLW   '8'                 ; ASCII value of "8" in W register
040F 3439           00610                RETLW   '9'                 ; ASCII value of "9" in W register
                    00611                                            ; Any enter after is in error (Display an E)
0410 3445           00612                RETLW   'E'                 ; ASCII value of "E" in W register
0411 3445           00613                RETLW   'E'                 ; ASCII value of "E" in W register
0412 3445           00614                RETLW   'E'                 ; ASCII value of "E" in W register
0413 3445           00615                RETLW   'E'                 ; ASCII value of "E" in W register
0414 3445           00616                RETLW   'E'                 ; ASCII value of "E" in W register
0415 3445           00617 NUM_TBL_END    RETLW   'E'                 ; ASCII value of "E" in W register
                    00618 ;
                    00619     if ( (NUM_TBL & 0xFF00) != (NUM_TBL_END & 0xFF00) )
                    00620        MESSG   "Warning: Table NUM_TBL crosses page boundry in computed jump"
                    00621     endif
                    00622 ;
                    00623 ;
07FF                00624     org     PMEM_END                       ; End of Program Memory
07FF 28D0           00625             GOTO    ERROR1                 ; If you get here your program was lost
                    00626
                    00627     end
```

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

```
0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX X--------------- ----------------
0400 : XXXXXXXXXXXXXXXX XXXXXX---------- ---------------- ----------------
07C0 : ---------------- ---------------- ---------------- ---------------X
```

© 1997 Microchip Technology Inc.

```
All other memory blocks unused.

Program Memory Words Used:   376
Program Memory Words Free:  3720


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,    16 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: CLOCK_01.H INCLUDE FILE

```
   nolist
;*****************************************************************************
;
;  This is the custom Header File for the real time clock application note
;  PROGRAM: CLOCK.H
;  Revision:5-10-94
;
;*****************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq          EQU   D'4000000'      ; Device Frequency is 4 MHz
DB_HI_BYTE        EQU  (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY    EQU  (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR        EQU  40  RAM Location
OUTER_CNTR        EQU  41               ; RAM Location
;
T1OSO             EQU 0                 ; The RC0 / T1OSO / T1CKI
;
RESET_V           EQU  0x0000    ; Address of RESET Vector
ISR_V             EQU  0x0004    ; Address of Interrupt Vector
PMEM_END          EQU  0x07FF    ; Last address in Program Memory
TABLE_ADDR        EQU  0x0400    ; Address where to start Tables
;
HR_MIN_SW         EQU  0x7       ; The switch to select the units
INC_SW            EQU  0x6       ; The switch to increment the selected units
CLR_MIN_SW        EQU  0x5       ; The switch to clear the minutes and seconds
;
FLAG_REG          EQU  0x020     ; Register which contains flag bits
;
;  +------+-------+-------+-----------+-------+-------+----------+---------+
;  |  AM  |  ---  |  ---  | KEY_INPUT |  ---  |  ---  | MIN_UNIT | HR_UNIT |
;  +------+-------+-------+-----------+-------+-------+----------+---------+
;
AM                EQU  0x07     ; Flag to specify if AM or PM
;
KEY_INPUT         EQU  0x04     ; Flag to specify if doing key inputs
;
MIN_UNIT          EQU  0x01     ; Flags to specify which units to operate on
HR_UNIT           EQU  0x00     ;  (HRS, MIN, or none)
;
HRS               EQU  0x030    ; Holds counter value for HOURS
MIN               EQU  0x031    ; Holds counter value for MINUTES
SECS              EQU  0x032    ; Holds counter value for SECONDS
MSD               EQU  0x033    ; Temporary register, Holds MSD of BIN to BCD conversion
LSD               EQU  0x034    ; Temporary register, Holds LSD of BIN to BCD conversion
TEMP              EQU  0x035    ; Temporary register
CHAR              EQU  0x036    ; Temporary register, Holds value to send to LCD module.
;
WAIT_CNTR         EQU  0x040    ; Counter that holds wait time for key inputs
;
;
; LCD Display Commands and Control Signal names.
;
E                 EQU  0        ; LCD Enable control line
R_W               EQU  1        ; LCD Read/Write control line
RS                EQU  2        ; LCD Register Select control line
;
;
; LCD Module commands
;
```

```
DISP_ON         EQU  0x00C   ; Display on
DISP_ON_C       EQU  0x00E   ; Display on, Cursor on
DISP_ON_B       EQU  0x00F   ; Display on, Cursor on, Blink cursor
DISP_OFF        EQU  0x008   ; Display off
CLR_DISP        EQU  0x001   ; Clear the Display
ENTRY_INC       EQU  0x006   ;
ENTRY_INC_S     EQU  0x007   ;
ENTRY_DEC       EQU  0x004   ;
ENTRY_DEC_S     EQU  0x005   ;
DD_RAM_ADDR     EQU  0x080   ; Least Significant 7-bit are for address
DD_RAM_UL       EQU  0x080   ; Upper Left corner of the Display
;

    list
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C: C74_REG.H INCLUDE FILE

```
    NOLIST
;
;   File =   C64_reg.h
;   Rev. History:      08-04-93 by MP
;                      10-18-93 by MP to make Page ok
;                      11-15-93 by MP to have correct pages for SFR
;
; EQUates for Special Function Registers
;
;
INDF            EQU     00
TMR0            EQU     01
OPTION_R        EQU     81
PCL             EQU     02
STATUS          EQU     03
FSR             EQU     04
PORTA           EQU     05
TRISA           EQU     85
PORTB           EQU     06
TRISB           EQU     86
PORTC           EQU     07
TRISC           EQU     87
PORTD           EQU     08
TRISD           EQU     88
PORTE           EQU     09
TRISE           EQU     89
PCLATH          EQU     0A
INTCON          EQU     0B
PIR1            EQU     0C
PIE1            EQU     8C
TMR1L           EQU     0E
PCON            EQU     8E
TMR1H           EQU     0F
T1CON           EQU     10
TMR2            EQU     11
T2CON           EQU     12
PR2             EQU     92
SSPBUF          EQU     13
SSPADD          EQU     93
SSPCON          EQU     14
SSPSTAT         EQU     94
CCPR1L          EQU     15
CCPR1H          EQU     16
CCP1CON         EQU     17
RCSTA           EQU     18
TXSTA           EQU     98
TXREG           EQU     19
SPBRG           EQU     99
RCREG           EQU     1A
CCPR2L          EQU     1B
CCPR2H          EQU     1C
CCP2CON         EQU     1D
ADRES           EQU     1E
ADCON0          EQU     1F
ADCON1          EQU     9F
;
;*********************************************
;************   Bit Definitions   ************
```


© 1997 Microchip Technology Inc.

```
;*************************************************
;
; STATUS register (Address 03/83)
;
IRP             EQU     7
RP1             EQU     6
RP0             EQU     5
TO              EQU     4
PD              EQU     3
Z               EQU     2
DC              EQU     1
C               EQU     0
;
; INTCON register (Address 0B/8B)
;
GI              EQU     7
PEIE            EQU     6
T0IE            EQU     5
INTE            EQU     4
RBIE            EQU     3
T0IF            EQU     2
INTF            EQU     1
RBIF            EQU     0
;
; PIR1 register (Address 0C)
;
PSPIF           EQU     7
SSPIF           EQU     3
CCP1IF          EQU     2
TMR2IF          EQU     1
TMR1IF          EQU     0
;
; PIE1 register (Address 8C)
;
PSPIE           EQU     7
SSPIE           EQU     3
CCP1IE          EQU     2
TMR2IE          EQU     1
TMR1IE          EQU     0
;
; OPTION register (Address 81)
;
RBPU            EQU     7
INTEDG          EQU     6
T0CS            EQU     5
T0SE            EQU     4
PSA             EQU     3
PS2             EQU     2
PS1             EQU     1
PS0             EQU     0
;
; PCON register (Address 8E)
;
POR             EQU     1
;
; TRISE register (Address 89)
;
IBF             EQU     7
OBF             EQU     6
IBOV            EQU     5
PSPMODE         EQU     4
TRISE2          EQU     2
TRISE1          EQU     1
TRISE0          EQU     0
;
; T1CON register (Address 10)
```


© 1997 Microchip Technology Inc.

```
;
T1CKPS1         EQU      5
T1CKPS0         EQU      4
T1OSCEN         EQU      3
T1INSYNC        EQU      2
TMR1CS          EQU      1
TMR1ON          EQU     0
;
; T2CON register (Address 12)
;
TOUTPS3         EQU     6
TOUTPS2         EQU     5
TOUTPS1         EQU     4
TOUTPS0         EQU     3
TMR2ON          EQU     2
T2CKPS1         EQU     1
T2CKPS0         EQU     0
;
; SSPCON register (Address 14)
;
WCOL            EQU     7
SSPOV           EQU     6
SSPEN           EQU     5
CKP             EQU     4
SSPM3           EQU     3
SSPM2           EQU     2
SSPM1           EQU     1
SSPM0           EQU     0
;
; SSPSTAT register (Address 94)
;
DA              EQU     5
P               EQU     4
S               EQU     3
RW              EQU     2
UA              EQU     1
BF              EQU     0
;
; CCP1CON register (Address 17)
;
CCP1X           EQU     5
CCP1Y           EQU     4
CCP1M3          EQU     3
CCP1M2          EQU     2
CCP1M1          EQU     1
CCP1M0          EQU     0
;
; RCSTA register (Address 18)
;
SPEN           EQU      7
RC89           EQU      6
SREN           EQU      5
CREN           EQU      4
FERR           EQU      2
OERR           EQU      1
RCD8           EQU      0
;
; TXSTA register (Address 98)
;
CSRC           EQU      7
TX89           EQU      6
TXEN           EQU      5
SYNC           EQU      4
BRGH           EQU      2
TRMT           EQU      1
TXD8           EQU      0
```


© 1997 Microchip Technology Inc.

```
;
; CCP2CON register (Address 1D)
;
CCP2X           EQU     5
CCP2Y           EQU     4
CCP2M3          EQU     3
CCP2M2          EQU     2
CCP2M1          EQU     1
CCP2M0          EQU     0
;
; ADCON0 register (Address 1F)
;
ADCS1           EQU     7
ADCS0           EQU     6
CHS2            EQU     5
CHS1            EQU     4
CHS0            EQU     3
GO              EQU     2
DONE            EQU     2
ADON            EQU     0
;
; ADCON1 register (Address 9F)
;
PCFG2           EQU     2
PCFG1           EQU     1
PCFG0           EQU     0
;
;***********************************************
;**** Bits for destination control
;****    W = W register is destination
;****    F = File register is destination
;***********************************************
;
W               EQU     0
F               EQU     1
;
FALSE           EQU     0
TRUE            EQU     1

    LIST
```


© 1997 Microchip Technology Inc.

AN587

Interfacing PICmicros™ to an LCD Module

Author: *Mark Palmer*
Code: *Mark Palmer/Scott Fink*
Microchip Technology Inc.

INTRODUCTION

This application note interfaces a micrange PICmicro device to a Hitachi® LM032L LCD character display module, with a two line by twenty character display. LCD modules are useful for displaying text information from a system. In large volume applications, the use of custom LCD displays becomes economical. The routines provided should be a good starting point for users whose applications implement a custom LCD. This source code should be compatible with the PIC16C5X devices, after modifications for the special function register initialization, but has not been verified on those devices.

OPERATION

The Hitachi LM032L LCD character display module can operate in one of two modes. The first (and default) mode is the 4-bit data interface mode. The second is the 8-bit data interface mode. When operating in 4-bit mode, two transfers per character / command are required. 8-bit mode, though easier to implement (less program memory) requires four additional I/O lines. The use of 8-bit mode is strictly a program memory size vs. I/O trade-off. The three most common data interfaces from the microcontroller are:

1. An 8-bit interface.
2. A 4-bit interface, with data transfers on the high nibble of the port.
3. A 4-bit interface, with data transfers on the low nibble of the port.

The LCD module also has three control signals, Enable (E), Read/Write (R_W), and Register Select (RS). The function of each control signal is shown in Table 1.

TABLE 1: CONTROL SIGNAL FUNCTIONS

Control Signal	Function
E	Causes data/control state to be latched Rising Edge = Latches control state (RS and R_W) Falling Edge = Latches data
RS	Register Select Control 1 = LCD in data mode 0 = LCD in command mode
R_W	Read / Write control 1 = LCD to write data 0 = LCD to read data

A single source file, with conditional assembly is used to generate each of these three options. This requires two flags. The flags and their results are shown in Table 2.

TABLE 2: CONDITIONAL ASSEMBLY FLAGS

Flags		
Four_bit	Data_HI	Result
1	0	4-bit mode. Data transferred on the low nibble of the port.
1	1	4-bit mode. Data transferred on the high nibble of the port.
0	x	8-bit mode.

© 1997 Microchip Technology Inc.

Figure 1, Figure 2, and Figure 3 show the block diagrams for the three different data interfaces. The LCD_CNTL and LCD_DATA lines are user definable to their port assignment. This is accomplished with EQUate statements in the source code. See Appendices B, C, and D.

FIGURE 1: 8-BIT DATA INTERFACE

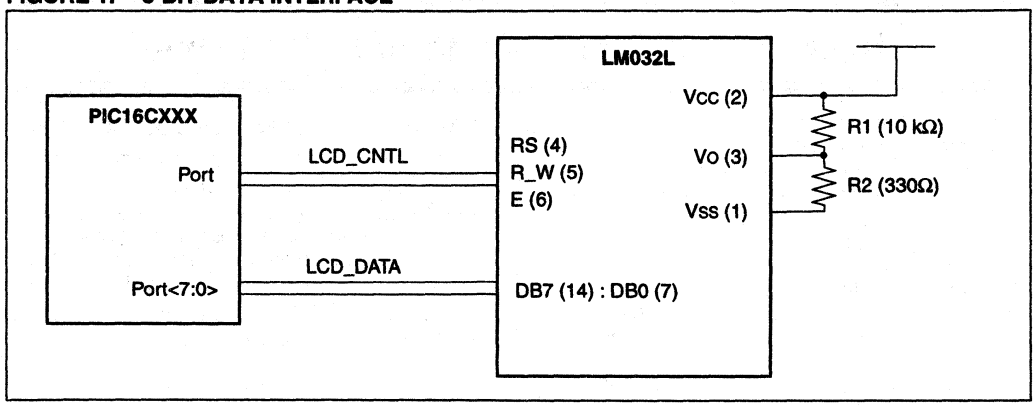


FIGURE 2: 4-BIT MODE; DATA TRANSFERRED ON THE HIGH NIBBLE OF THE PORT

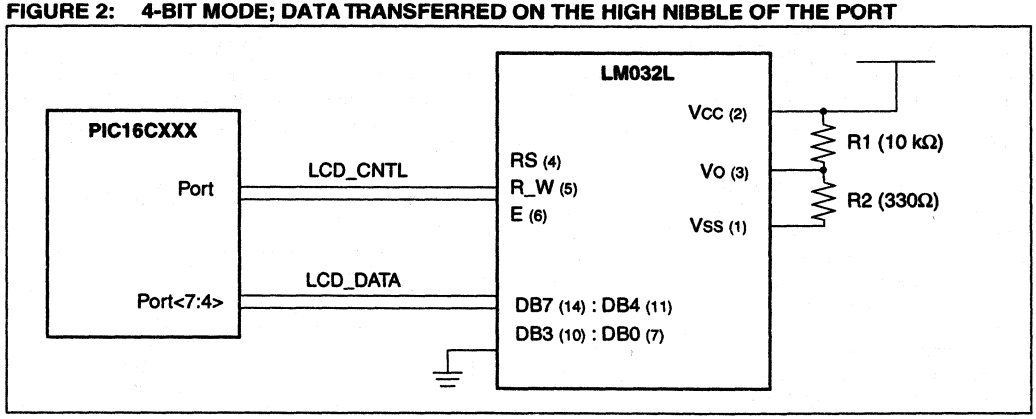
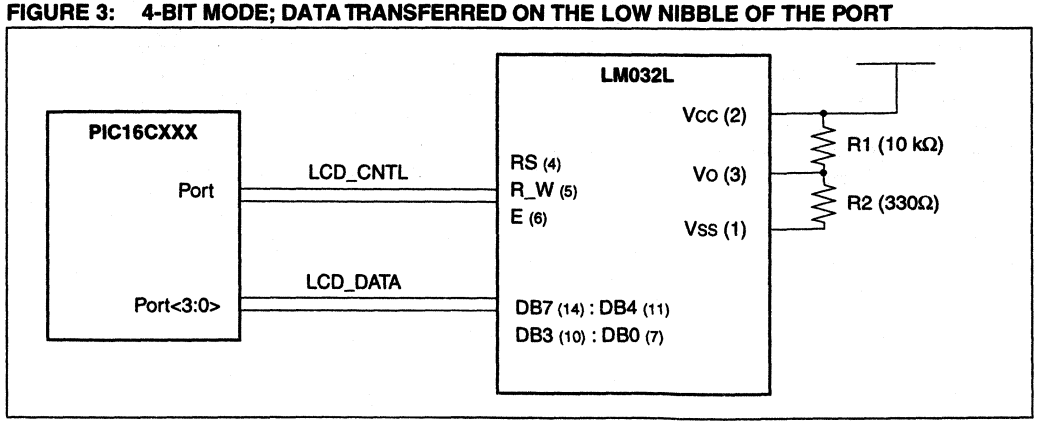


FIGURE 3: 4-BIT MODE; DATA TRANSFERRED ON THE LOW NIBBLE OF THE PORT



© 1997 Microchip Technology Inc.

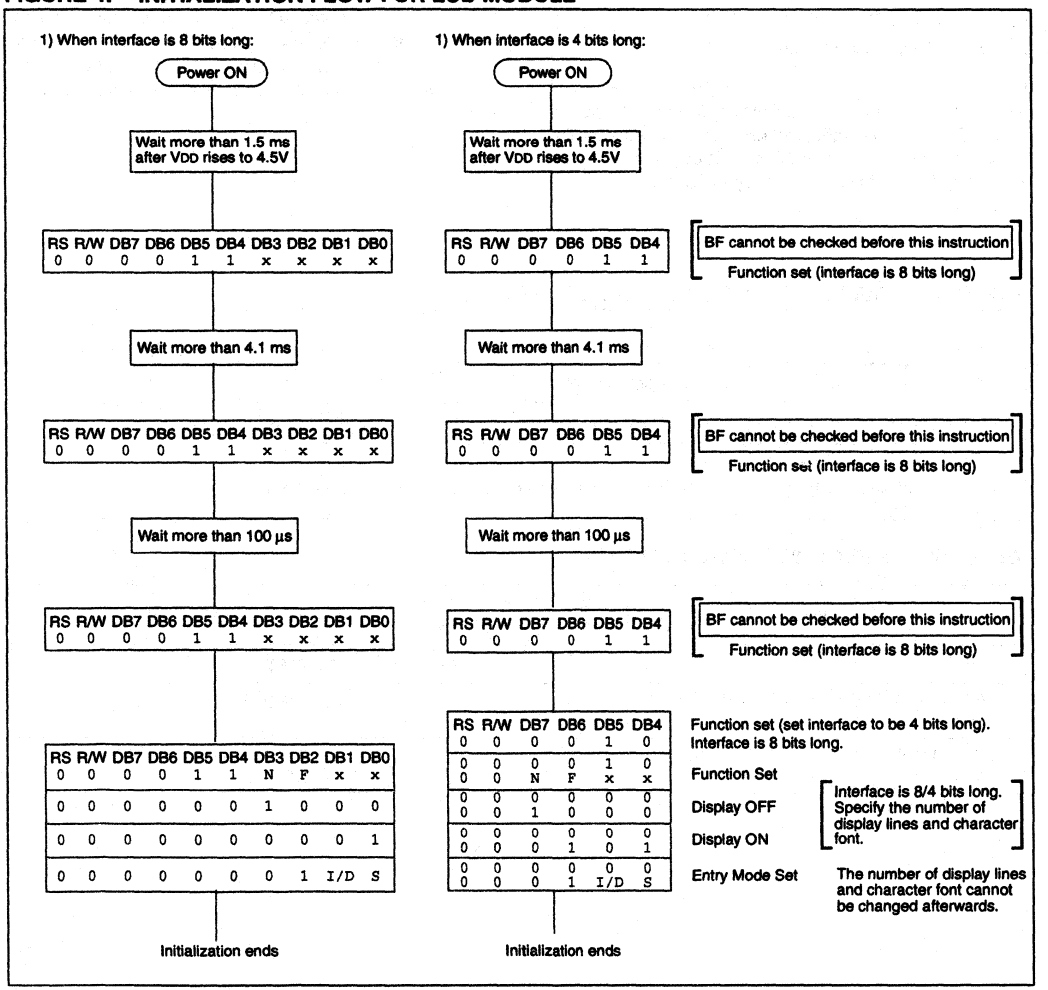
LCD's (drivers) are slow devices when compared to microcontrollers. Care must be taken from having communication occur too quickly. The software will need to control communicaton speed and timing to ensure the slow LCD and fast microcontroller can stay synchronized. The timing requirements of the LM032L are shown in Appendix A. We recommend that the complete specifications of the LM032L be acquired from Hitachi or a Hitachi distributor. The literature numbers are CE-E613Q and M24T013 for a LM032L display driver.

When the module powers up, the default data transfer mode is 8-bit. The initialization sequence only requires commands that are 4-bit in length. The last initialization command needs to specify the data transfer width (4-or 8-bit). Then a delay of 4.6 ms must be executed before the LCD module can be initialized. Some of the LCD module commands are:

- 1 or 2 lines of characters
- Display on /off
- Clear display
- Increment / do not increment character address pointer after each character
- Load character address pointer

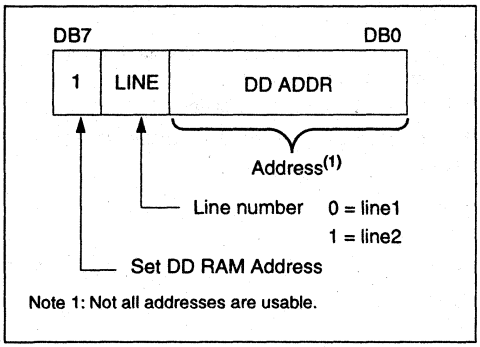
The initialization flow for the module is shown in Figure 4.

FIGURE 4: INITIALIZATION FLOW FOR LCD MODULE



After initialization, each character address is individually addressable. Figure 5 shows the structure of the command to specify the character address.

FIGURE 5: CHARACTER ADDRESS COMMAND FORMAT



The Hitachi Display Drive (HD44780A) has 80 bytes of RAM. The LM032L modules only use 40 bytes of the available RAM (2 x 20 characters). It is possible to use the remaining RAM locations for storage of other information.

Figure 6 shows the display data positions supported by the display driver as well as the characters actually displayed by the module (the non-shaded addresses).

The program example implemented here uses the character auto increment feature. This automatically increments the character address pointer after each character is written to the display.

CONCLUSION

The Hitachi LM032L character display module is well suited for displaying information. The selection of 4-bit or 8-bit data transfer mode is strictly a program memory size vs. I/O resource trade-off. The supplied code is easily used in any of three common data interfaces. The source is easily modifiable to a designers specific application needs. Other display modules/drivers maybe implemented with the appropriate modifications. Table 3 shows the resource requirements for the three subroutines `SEND_CHAR`, `SEND_COMMAND`, and `BUSY_CHECK` in the various data interface modes.

FIGURE 6: DISPLAY DRIVER (DD) RAM LOCATIONS

digit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	- - - -	33	34	35	36	37	38	39	40	← Display position
line-1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	- - - -	20	21	22	23	24	25	26	27	← DD RAM address (Hexadecimal)
line-2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	- - - -	60	61	62	63	64	65	66	67	

Note: Shaded locations are not displayed on the LM032L display module.

TABLE 3: RESOURCE REQUIREMENTS

Mode	Program Memory	Data Memory	Verified On
8-bit	32	3	PICDEM-2(1)
4-bit, Data transferred on the high nibble of the port.	53	3	PICDEM-2(1)
4-bit, Data transferred on the high nibble of the port.	53	3	Low-Power Real-Time Clock Board (AN582)

Note 1: Jumper J6 must be removed.

© 1997 Microchip Technology Inc.

APPENDIX A: LM032L TIMING REQUIREMENTS

TABLE A-1: TIMING CHARACTERISTICS

Parameter #	Symbol	Characteristics	Min.	Typ.	Max.	Unit
1	TCYC	Enable cycle time	1.0	—	—	µs
2	PWEH	Enable pulse width	450	—	—	µs
3	TER, TEF	Enable rise / fall time	—	—	25	µs
4	TAS	RS, R/W set-up time	140	—	—	µs
5	TDDR	Data delay time	—	—	320	µs
6	TDSU	Data setup time	195	—	—	µs
7	TH	Hold time	20	—	—	µs

FIGURE A-1: DATA WRITE INTERFACE TIMING

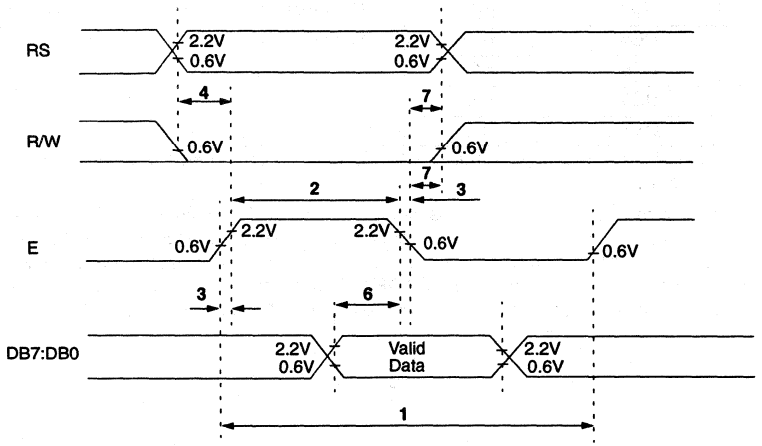
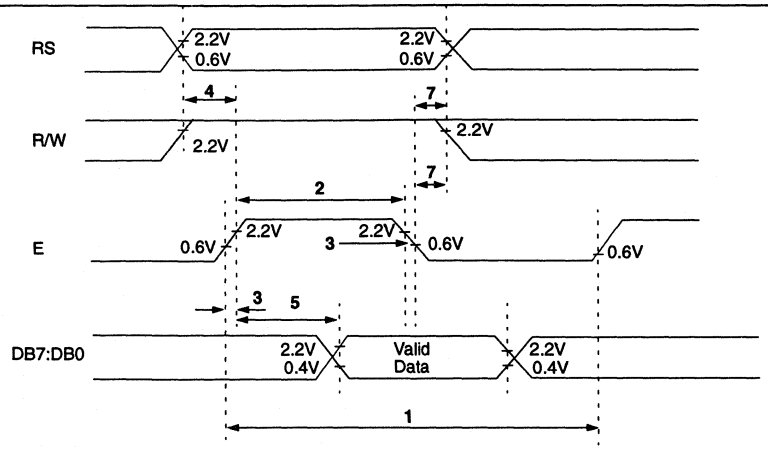


FIGURE A-2: DATA READ INTERFACE TIMING



Note: Refer to Hitachi documentation for the most current timing specifications.

TABLE A-2: LM032L PIN CONNECTION

Pin No.	Symbol	Level	Function	
1	V_{SS}	—	0V	Ground
2	V_{DD}	—	+5V	Power Supply(+)
3	V_O	—	—	Ground
4	RS	H/L	L: Instruction Code Input H: Data Input	
5	R/W	H/L	H: Data Read (LCD module→MPU) L: Data Write (LCD module←MPU)	
6	E	H,H→L	Enable Signal	
7	DB0	H/L	Data Bus Line Note (1), (2)	
8	DB1	H/L		
9	DB2	H/L		
10	DB3	H/L		
11	DB4	H/L		
12	DB5	H/L		
13	DB6	H/L		
14	DB7	H/L		

In the HD44780, the data can be sent in either two 4-bit operations or one 8-bit operation, This flexibility allows an interface to both 4- and 8-bit MPUs.

Note 1: When interface data is 4-bits long, data is transferred using only 4 lines of DB7:DB4 (DB3:DB0 are not used). Data transfer between the HD44780 and the MPU completes when 4-bits of data is transferred twice. Data of the higher order 4 bits (contents of DB7:DB4 when interface data is 8-bits long) is transferred first and then lower order 4 bits (contents of DB3:DB0 when interface data is 8-bits long).

2: When interface data is 8-bits long, data is transferred using 8 data lines of DB7:DB0.

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: 8-BIT DATA INTERFACE LISTING

```
MPASM 01.40.01 Intermediate    LM032L.ASM   4-7-1997  9:43:02         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST P=16C64
                     00002         ERRORLEVEL  -302
                     00003 ;
                     00004 ; This program interfaces to a Hitachi (LM032L) 2 line by 20 character display
                     00005 ; module. The program assembles for either 4-bit or 8-bit data interface, depending
                     00006 ; on the value of the 4bit flag. LCD_DATA is the port which supplies the data to
                     00007 ; the LM032L, while LCD_CNTL is the port that has the control lines ( E, RS, RW ).
                     00008 ; In 4-bit mode the data is transfer on the high nibble of the port ( PORT<7:4> ).
                     00009 ;
                     00010 ;       Program = LM032L.ASM
                     00011 ;       Revision Date:   5-10-94
                     00012 ;                        1-22-97    Compatibility with MPASMWIN 1.40
                     00013 ;
                     00014 ;
                     00015        include <p16c64.inc>
                     00001         LIST
                     00002 ; P16C64.INC Standard Header File, Version 1.01 Microchip Technology, Inc.
                     00238         LIST
                     00016
  0000009F           00017 ADCON1           EQU  9F
                     00018
  00000000           00019 FALSE            EQU  0
  00000001           00020 TRUE             EQU  1
                     00021
                     00022        include <lm032l.h>
                     00069         list
                     00023 ;
  00000001           00024 Four_bit         EQU  TRUE              ; Selects 4- or 8-bit data transfers
  00000000           00025 Data_HI          EQU  FALSE             ; If 4-bit transfers, Hi or Low nibble of PORT
                     00026 ;
                     00027 ;
                     00028     if ( Four_bit && !Data_HI )
                     00029 ;
  00000006           00030 LCD_DATA         EQU  PORTB
  00000086           00031 LCD_DATA_TRIS    EQU  TRISB
```

```
                        00032 ;
                        00033     else
                        00034 ;
                        00035 LCD_DATA          EQU  PORTD
                        00036 LCD_DATA_TRIS     EQU  TRISD
                        00037 ;
                        00038     endif
                        00039 ;
  00000005              00040 LCD_CNTL          EQU  PORTA
                        00041 ;
                        00042 ;
                        00043 ;
                        00044 ; LCD Display Commands and Control Signal names.
                        00045 ;
                        00046     if ( Four_bit && !Data_HI )
                        00047 ;
  00000000              00048 E                 EQU  0                  ; LCD Enable control line
  00000001              00049 RW                EQU  1                  ; LCD Read/Write control line
  00000002              00050 RS                EQU  2                  ; LCD Register Select control line
                        00051 ;
                        00052     else
                        00053 ;
                        00054 E                 EQU  3                  ; LCD Enable control line
                        00055 RW                EQU  2                  ; LCD Read/Write control line
                        00056 RS                EQU  1                  ; LCD Register Select control line
                        00057 ;
                        00058     endif
                        00059 ;
                        00060 ;
  00000030              00061 TEMP1             EQU  0x030
                        00062 ;
0000                    00063         org       RESET_V                 ; RESET vector location
0000 2808               00064 RESET       GOTO    START                   ;
                        00065 ;
                        00066 ; This is the Periperal Interrupt routine. Should NOT get here
                        00067 ;
                        00068     page
0004                    00069         org       ISR_V                   ; Interrupt vector location
0004                    00070 PER_INT_V
0004 1283               00071 ERROR1            BCF  STATUS, RP0        ; Bank 0
0005 1407               00072                   BSF  PORTC, 0
0006 1007               00073                   BCF  PORTC, 0
0007 2804               00074                   GOTO ERROR1
                        00075 ;
                        00076 ;
                        00077 ;
0008                    00078 START                                     ; POWER_ON Reset (Beginning of program)
```


© 1997 Microchip Technology Inc.

```
0008 0183          00079             CLRF    STATUS                  ; Do initialization (Bank 0)
0009 018B          00080             CLRF    INTCON
000A 018C          00081             CLRF    PIR1
000B 1683          00082             BSF     STATUS, RP0             ; Bank 1
000C 3000          00083             MOVLW   0x00                    ; The LCD module does not like to work w/ weak pull-ups
000D 0081          00084             MOVWF   OPTION_REG              ;
000E 018C          00085             CLRF    PIE1                    ; Disable all peripheral interrupts
                   00086 ;***
                   00087 ;***   If using device with A/D, these two instructions are required.
                   00088 ;***
                   00089 ;           MOVLW   0xFF                    ;
                   00090 ;           MOVWF   ADCON1                  ; Port A is Digital.
                   00091 ;
                   00092 ;
000F 1283          00093             BCF     STATUS, RP0             ; Bank 0
0010 0185          00094             CLRF    PORTA                   ; ALL PORT output should output Low.
0011 0186          00095             CLRF    PORTB
0012 0187          00096             CLRF    PORTC
0013 0188          00097             CLRF    PORTD
0014 0189          00098             CLRF    PORTE
0015 1010          00099             BCF     T1CON, TMR1ON           ; Timer 1 is NOT incrementing
                   00100 ;
0016 1683          00101             BSF     STATUS, RP0             ; Select Bank 1
0017 0185          00102             CLRF    TRISA                   ; RA5 -  0 outputs
0018 30F0          00103             MOVLW   0xF0                    ;
0019 0086          00104             MOVWF   TRISB                   ; RB7 - 4 inputs, RB3 - 0 outputs
001A 0187          00105             CLRF    TRISC                   ; RC Port are outputs
001B 1407          00106             BSF     TRISC, T1OSO            ; RC0 needs to be input for the oscillator to function
001C 0188          00107             CLRF    TRISD                   ; RD Port are outputs
001D 0189          00108             CLRF    TRISE                   ; RE Port are outputs
001E 140C          00109             BSF     PIE1, TMR1IE            ; Enable TMR1 Interrupt
001F 1781          00110             BSF     OPTION_REG,NOT_RBPU     ; Disable PORTB pull-ups
0020 1283          00111             BCF     STATUS, RP0             ; Select Bank 0
                   00112 ;
                   00113     page
                   00114 ;
                   00115 ; Initialize the LCD Display Module
                   00116 ;
0021 0185          00117             CLRF    LCD_CNTL                ; ALL PORT output should output Low.
                   00118
0022               00119 DISPLAY_INIT
                   00120     if ( Four_bit && !Data_HI )
0022 3002          00121             MOVLW   0x02                    ; Command for 4-bit interface low nibble
                   00122     endif
                   00123 ;
                   00124     if ( Four_bit && Data_HI )
                   00125             MOVLW   0x020                   ; Command for 4-bit interface high nibble
```

© 1997 Microchip Technology Inc.

```
                    00126     endif
                    00127 ;
                    00128     if ( !Four_bit )
                    00129             MOVLW   0x038               ; Command for 8-bit interface
                    00130     endif
                    00131 ;
0023 0086           00132             MOVWF   LCD_DATA            ;
0024 1405           00133             BSF     LCD_CNTL, E         ;
0025 1005           00134             BCF     LCD_CNTL, E         ;
                    00135 ;
                    00136 ; This routine takes the calculated times that the delay loop needs to
                    00137 ; be executed, based on the LCD_INIT_DELAY EQUate that includes the
                    00138 ; frequency of operation. These uses registers before they are needed to
                    00139 ; store the time.
                    00140 ;
0026 3006           00141 LCD_DELAY   MOVLW   LCD_INIT_DELAY      ;
0027 00B3           00142             MOVWF   MSD                 ; Use MSD and LSD Registers to Initialize LCD
0028 01B4           00143             CLRF    LSD                 ;
0029 0BB4           00144 LOOP2       DECFSZ  LSD, F              ; Delay time = MSD * ((3 * 256) + 3) * Tcy
002A 2829           00145             GOTO    LOOP2               ;
002B 0BB3           00146             DECFSZ  MSD, F              ;
002C                00147 END_LCD_DELAY
002C 2829           00148             GOTO    LOOP2               ;
                    00149 ;
                    00150 ; Command sequence for 2 lines of 5x7 characters
                    00151 ;
002D                00152 CMD_SEQ
                    00153 ;
                    00154     if ( Four_bit )
                    00155         if ( !Data_HI )
002D 3002           00156             MOVLW   0X02                ; 4-bit low nibble xfer
                    00157         else
                    00158             MOVLW   0X020               ; 4-bit high nibble xfer
                    00159         endif
                    00160 ;
                    00161     else                                ; 8-bit mode
                    00162             MOVLW   0X038
                    00163     endif
                    00164 ;
002E 0086           00165             MOVWF   LCD_DATA            ; This code for both 4-bit and 8-bit modes
002F 1405           00166             BSF     LCD_CNTL, E         ;
0030 1005           00167             BCF     LCD_CNTL, E         ;
                    00168 ;
                    00169     if ( Four_bit )                     ; This code for only 4-bit mode (2nd xfer)
                    00170         if ( !Data_HI )
0031 3008           00171             MOVLW   0x08                ; 4-bit low nibble xfer
                    00172         else
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

```
                    00173             MOVLW   0x080               ; 4-bit high nibble xfer
                    00174         endif
0032 0086           00175             MOVWF   LCD_DATA            ;
0033 1405           00176             BSF     LCD_CNTL, E         ;
0034 1005           00177             BCF     LCD_CNTL, E         ;
                    00178     endif
                    00179 ;
                    00180 ; Busy Flag should be valid after this point
                    00181 ;
0035 300C           00182             MOVLW   DISP_ON             ;
0036 2072           00183             CALL    SEND_CMD            ;
0037 3001           00184             MOVLW   CLR_DISP            ;
0038 2072           00185             CALL    SEND_CMD            ;
0039 3006           00186             MOVLW   ENTRY_INC           ;
003A 2072           00187             CALL    SEND_CMD            ;
003B 3080           00188             MOVLW   DD_RAM_ADDR         ;
003C 2072           00189             CALL    SEND_CMD            ;
                    00190 ;
                    00191     page
                    00192 ;
                    00193 ;Send a message the hard way
003D 304D           00194             movlw   'M'
003E 2063           00195             call    SEND_CHAR
003F 3069           00196             movlw   'i'
0040 2063           00197             call    SEND_CHAR
0041 3063           00198             movlw   'c'
0042 2063           00199             call    SEND_CHAR
0043 3072           00200             movlw   'r'
0044 2063           00201             call    SEND_CHAR
0045 306F           00202             movlw   'o'
0046 2063           00203             call    SEND_CHAR
0047 3063           00204             movlw   'c'
0048 2063           00205             call    SEND_CHAR
0049 3068           00206             movlw   'h'
004A 2063           00207             call    SEND_CHAR
004B 3069           00208             movlw   'i'
004C 2063           00209             call    SEND_CHAR
004D 3070           00210             movlw   'p'
004E 2063           00211             call    SEND_CHAR
                    00212
004F 30C0           00213             movlw   B'11000000'         ;Address DDRam first character, second line
0050 2072           00214             call    SEND_CMD
                    00215
                    00216                                         ;Demonstration of the use of a table to output a message
0051 3000           00217             movlw   0                   ;Table address of start of message
0052                00218 dispmsg
0052 00B0           00219             movwf   TEMP1               ;TEMP1 holds start of message address
```

```
0053 2099          00220             call    Table
0054 39FF          00221             andlw   0FFh                ;Check if at end of message (zero
0055 1903          00222             btfsc   STATUS,Z            ;returned at end)
0056 285B          00223             goto    out
0057 2063          00224             call    SEND_CHAR          ;Display character
0058 0830          00225             movf    TEMP1,w            ;Point to next character
0059 3E01          00226             addlw   1
005A 2852          00227             goto    dispmsg
005B               00228 out
005B               00229 loop
005B 285B          00230             goto    loop               ;Stay here forever
                   00231 ;
                   00232 ;
005C               00233 INIT_DISPLAY
005C 300C          00234             MOVLW   DISP_ON            ; Display On, Cursor On
005D 2072          00235             CALL    SEND_CMD           ; Send This command to the Display Module
005E 3001          00236             MOVLW   CLR_DISP           ; Clear the Display
005F 2072          00237             CALL    SEND_CMD           ; Send This command to the Display Module
0060 3006          00238             MOVLW   ENTRY_INC          ; Set Entry Mode Inc., No shift
0061 2072          00239             CALL    SEND_CMD           ; Send This command to the Display Module
0062 0008          00240             RETURN
                   00241 ;
                   00242     page
                   00243 ;
                   00244 ;*****************************************************************
                   00245 ;* The LCD Module Subroutines                                    *
                   00246 ;*****************************************************************
                   00247 ;
                   00248     if ( Four_bit )      ; 4-bit Data transfers?
                   00249 ;
                   00250         if ( Data_HI )   ; 4-bit transfers on the high nibble of the PORT
                   00251 ;
                   00252 ;*****************************************************************
                   00253 ;*SendChar - Sends character to LCD                            *
                   00254 ;*This routine splits the character into the upper and lower    *
                   00255 ;*nibbles and sends them to the LCD, upper nibble first.       *
                   00256 ;*****************************************************************
                   00257 ;
                   00258 SEND_CHAR
                   00259             MOVWF   CHAR               ;Character to be sent is in W
                   00260             CALL    BUSY_CHECK         ;Wait for LCD to be ready
                   00261             MOVF    CHAR, w
                   00262             ANDLW   0xF0               ;Get upper nibble
                   00263             MOVWF   LCD_DATA           ;Send data to LCD
                   00264             BCF     LCD_CNTL, RW       ;Set LCD to read
                   00265             BSF     LCD_CNTL, RS       ;Set LCD to data mode
                   00266             BSF     LCD_CNTL, E        ;toggle E for LCD
```


© 1997 Microchip Technology Inc.

```
                00267             BCF     LCD_CNTL, E
                00268             SWAPF   CHAR, w
                00269             ANDLW   0xF0            ;Get lower nibble
                00270             MOVWF   LCD_DATA        ;Send data to LCD
                00271             BSF     LCD_CNTL, E     ;toggle E for LCD
                00272             BCF     LCD_CNTL, E
                00273             RETURN
                00274 ;
                00275         else              ; 4-bit transfers on the low nibble of the PORT
                00276 ;
                00277 ;*****************************************************************
                00278 ;* SEND_CHAR - Sends character to LCD                            *
                00279 ;* This routine splits the character into the upper and lower    *
                00280 ;* nibbles and sends them to the LCD, upper nibble first.        *
                00281 ;* The data is transmitted on the PORT<3:0> pins                 *
                00282 ;*****************************************************************
                00283 ;
0063            00284 SEND_CHAR
0063 00B6       00285             MOVWF   CHAR            ; Character to be sent is in W
0064 2081       00286             CALL    BUSY_CHECK      ; Wait for LCD to be ready
0065 0E36       00287             SWAPF   CHAR, W
0066 390F       00288             ANDLW   0x0F            ; Get upper nibble
0067 0086       00289             MOVWF   LCD_DATA        ; Send data to LCD
0068 1085       00290             BCF     LCD_CNTL, RW    ; Set LCD to read
0069 1505       00291             BSF     LCD_CNTL, RS    ; Set LCD to data mode
006A 1405       00292             BSF     LCD_CNTL, E     ; toggle E for LCD
006B 1005       00293             BCF     LCD_CNTL, E
006C 0836       00294             MOVF    CHAR, W
006D 390F       00295             ANDLW   0x0F            ; Get lower nibble
006E 0086       00296             MOVWF   LCD_DATA        ; Send data to LCD
006F 1405       00297             BSF     LCD_CNTL, E     ; toggle E for LCD
0070 1005       00298             BCF     LCD_CNTL, E
0071 0008       00299             RETURN
                00300 ;
                00301         endif
                00302     else
                00303 ;
                00304 ;*****************************************************************
                00305 ;* SEND_CHAR - Sends character contained in register W to LCD    *
                00306 ;* This routine sends the entire character to the PORT           *
                00307 ;* The data is transmitted on the PORT<7:0> pins                 *
                00308 ;*****************************************************************
                00309 ;
                00310 SEND_CHAR
                00311             MOVWF   CHAR            ; Character to be sent is in W
                00312             CALL    BUSY_CHECK      ; Wait for LCD to be ready
                00313             MOVF    CHAR, w
```

© 1997 Microchip Technology Inc.

```
00314             MOVWF   LCD_DATA        ; Send data to LCD
00315             BCF     LCD_CNTL, RW    ; Set LCD in read mode
00316             BSF     LCD_CNTL, RS    ; Set LCD in data mode
00317             BSF     LCD_CNTL, E     ; toggle E for LCD
00318             BCF     LCD_CNTL, E
00319             RETURN
00320 ;
00321     endif
00322 ;
00323     page
00324 ;
00325 ;*********************************************************************
00326 ;* SendCmd - Sends command to LCD                                    *
00327 ;* This routine splits the command into the upper and lower          *
00328 ;* nibbles and sends them to the LCD, upper nibble first.            *
00329 ;* The data is transmitted on the PORT<3:0> pins                     *
00330 ;*********************************************************************
00331 ;
00332     if ( Four_bit )       ; 4-bit Data transfers?
00333 ;
00334         if ( Data_HI )    ; 4-bit transfers on the high nibble of the PORT
00335 ;
00336 ;*********************************************************************
00337 ;* SEND_CMD - Sends command to LCD                                   *
00338 ;* This routine splits the command into the upper and lower          *
00339 ;* nibbles and sends them to the LCD, upper nibble first.            *
00340 ;*********************************************************************
00341
00342 SEND_CMD
00343             MOVWF   CHAR            ; Character to be sent is in W
00344             CALL    BUSY_CHECK      ; Wait for LCD to be ready
00345             MOVF    CHAR,w
00346             ANDLW   0xF0            ; Get upper nibble
00347             MOVWF   LCD_DATA        ; Send data to LCD
00348             BCF     LCD_CNTL,RW     ; Set LCD to read
00349             BCF     LCD_CNTL,RS     ; Set LCD to command mode
00350             BSF     LCD_CNTL,E      ; toggle E for LCD
00351             BCF     LCD_CNTL,E
00352             SWAPF   CHAR,w
00353             ANDLW   0xF0            ; Get lower nibble
00354             MOVWF   LCD_DATA        ; Send data to LCD
00355             BSF     LCD_CNTL,E      ; toggle E for LCD
00356             BCF     LCD_CNTL,E
00357             RETURN
00358 ;
00359         else              ; 4-bit transfers on the low nibble of the PORT
00360 ;
```


© 1997 Microchip Technology Inc.

```
0072                00361 SEND_CMD
0072 00B6           00362             MOVWF   CHAR            ; Character to be sent is in W
0073 2081           00363             CALL    BUSY_CHECK      ; Wait for LCD to be ready
0074 0E36           00364             SWAPF   CHAR, W
0075 390F           00365             ANDLW   0x0F            ; Get upper nibble
0076 0086           00366             MOVWF   LCD_DATA        ; Send data to LCD
0077 1085           00367             BCF     LCD_CNTL, RW    ; Set LCD to read
0078 1105           00368             BCF     LCD_CNTL, RS    ; Set LCD to command mode
0079 1405           00369             BSF     LCD_CNTL, E     ; toggle E for LCD
007A 1005           00370             BCF     LCD_CNTL, E
007B 0836           00371             MOVF    CHAR, W
007C 390F           00372             ANDLW   0x0F            ; Get lower nibble
007D 0086           00373             MOVWF   LCD_DATA        ; Send data to LCD
007E 1405           00374             BSF     LCD_CNTL, E     ; toggle E for LCD
007F 1005           00375             BCF     LCD_CNTL, E
0080 0008           00376             RETURN
                    00377 ;
                    00378         endif
                    00379     else
                    00380 ;
                    00381 ;*****************************************************************
                    00382 ;* SEND_CND - Sends command contained in register W to LCD     *
                    00383 ;* This routine sends the entire character to the PORT         *
                    00384 ;* The data is transmitted on the PORT<7:0> pins               *
                    00385 ;*****************************************************************
                    00386
                    00387 SEND_CMD
                    00388             MOVWF   CHAR            ; Command to be sent is in W
                    00389             CALL    BUSY_CHECK      ; Wait for LCD to be ready
                    00390             MOVF    CHAR, w
                    00391             MOVWF   LCD_DATA        ; Send data to LCD
                    00392             BCF     LCD_CNTL, RW    ; Set LCD in read mode
                    00393             BCF     LCD_CNTL, RS    ; Set LCD in command mode
                    00394             BSF     LCD_CNTL, E     ; toggle E for LCD
                    00395             BCF     LCD_CNTL, E
                    00396             RETURN
                    00397 ;
                    00398     endif
                    00399 ;
                    00400     page
                    00401 ;
                    00402     if ( Four_bit )                 ; 4-bit Data transfers?
                    00403 ;
                    00404         if ( Data_HI )              ; 4-bit transfers on the high nibble of the PORT
                    00405 ;
                    00406 ;*****************************************************************
                    00407 ;* This routine checks the busy flag, returns when not busy      *
```

© 1997 Microchip Technology Inc.

```
00408 ;*  Affects:                                                        *
00409 ;*       TEMP - Returned with busy/address                         *
00410 ;*******************************************************************
00411 ;
00412 BUSY_CHECK
00413             BSF     STATUS, RP0     ; Select Register Bank1
00414             MOVLW   0xFF            ; Set Port_D for input
00415             MOVWF   LCD_DATA_TRIS
00416             BCF     STATUS, RP0     ; Select Register Bank0
00417             BCF     LCD_CNTL, RS    ; Set LCD for Command mode
00418             BSF     LCD_CNTL, RW    ; Setup to read busy flag
00419             BSF     LCD_CNTL, E     ; Set E high
00420             BCF     LCD_CNTL, E     ; Set E low
00421             MOVF    LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
00422             ANDLW   0xF0            ; Mask out lower nibble
00423             MOVWF   TEMP
00424             BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
00425             BCF     LCD_CNTL, E
00426             SWAPF   LCD_DATA, w     ; Read lower nibble busy flag, DDRam address
00427             ANDLW   0x0F            ; Mask out upper nibble
00428             IORWF   TEMP            ; Combine nibbles
00429             BTFSC   TEMP, 7         ; Check busy flag, high = busy
00430             GOTO    BUSY_CHECK      ; If busy, check again
00431             BCF     LCD_CNTL, RW
00432             BSF     STATUS, RP0     ; Select Register Bank1
00433             MOVLW   0x0F
00434             MOVWF   LCD_DATA_TRIS   ; Set Port_D for output
00435             BCF     STATUS, RP0     ; Select Register Bank0
00436             RETURN
00437 ;
00438         else                        ; 4-bit transfers on the low nibble of the PORT
00439 ;
00440 ;*******************************************************************
00441 ;* This routine checks the busy flag, returns when not busy        *
00442 ;*  Affects:                                                       *
00443 ;*       TEMP - Returned with busy/address                         *
00444 ;*******************************************************************
00445 ;
0081              00446 BUSY_CHECK
0081 1683         00447             BSF     STATUS, RP0     ; Bank 1
0082 30FF         00448             MOVLW   0xFF            ; Set PortB for input
0083 0086         00449             MOVWF   LCD_DATA_TRIS
0084 1283         00450             BCF     STATUS, RP0     ; Bank 0
0085 1105         00451             BCF     LCD_CNTL, RS    ; Set LCD for Command mode
0086 1485         00452             BSF     LCD_CNTL, RW    ; Setup to read busy flag
0087 1405         00453             BSF     LCD_CNTL, E     ; Set E high
0088 1005         00454             BCF     LCD_CNTL, E     ; Set E low
```


© 1997 Microchip Technology Inc.

```
0089 0E06           00455             SWAPF   LCD_DATA, W     ; Read upper nibble busy flag, DDRam address
008A 39F0           00456             ANDLW   0xF0            ; Mask out lower nibble
008B 00B5           00457             MOVWF   TEMP            ;
008C 1405           00458             BSF     LCD_CNTL, E     ; Toggle E to get lower nibble
008D 1005           00459             BCF     LCD_CNTL, E
008E 0806           00460             MOVF    LCD_DATA, W     ; Read lower nibble busy flag, DDRam address
008F 390F           00461             ANDLW   0x0F            ; Mask out upper nibble
0090 04B5           00462             IORWF   TEMP, F         ; Combine nibbles
0091 1BB5           00463             BTFSC   TEMP, 7         ; Check busy flag, high = busy
0092 2881           00464             GOTO    BUSY_CHECK      ; If busy, check again
0093 1085           00465             BCF     LCD_CNTL, RW
0094 1683           00466             BSF     STATUS, RP0     ; Bank 1
0095 30F0           00467             MOVLW   0xF0            ;
0096 0086           00468             MOVWF   LCD_DATA_TRIS   ; RB7 - 4 = inputs, RB3 - 0 = output
0097 1283           00469             BCF     STATUS, RP0     ; Bank 0
0098 0008           00470             RETURN
                    00471 ;
                    00472         endif
                    00473     else
                    00474 ;
                    00475 ;*****************************************************************
                    00476 ;* This routine checks the busy flag, returns when not busy    *
                    00477 ;*  Affects:                                                     *
                    00478 ;*       TEMP - Returned with busy/address                      *
                    00479 ;*****************************************************************
                    00480 ;
                    00481 BUSY_CHECK
                    00482             BSF     STATUS,RP0      ; Select Register Bank1
                    00483             MOVLW   0xFF            ; Set port_D for input
                    00484             MOVWF   LCD_DATA_TRIS
                    00485             BCF     STATUS, RP0     ; Select Register Bank0
                    00486             BCF     LCD_CNTL, RS    ; Set LCD for command mode
                    00487             BSF     LCD_CNTL, RW    ; Setup to read busy flag
                    00488             BSF     LCD_CNTL, E     ; Set E high
                    00489             BCF     LCD_CNTL, E     ; Set E low
                    00490             MOVF    LCD_DATA, w     ; Read busy flag, DDram address
                    00491             MOVWF   TEMP
                    00492             BTFSC   TEMP, 7         ; Check busy flag, high=busy
                    00493             GOTO    BUSY_CHECK
                    00494             BCF     LCD_CNTL, RW
                    00495             BSF     STATUS, RP0     ; Select Register Bank1
                    00496             MOVLW   0x00
                    00497             MOVWF   LCD_DATA_TRIS   ; Set port_D for output
                    00498             BCF     STATUS, RP0     ; Select Register Bank0
                    00499             RETURN
                    00500 ;
                    00501     endif
```

© 1997 Microchip Technology Inc.

```
                  00502     page
                  00503 ;
0099              00504 Table
0099 0782         00505             addwf   PCL, F          ;Jump to char pointed to in W reg
009A 344D         00506             retlw   'M'
009B 3469         00507             retlw   'i'
009C 3463         00508             retlw   'c'
009D 3472         00509             retlw   'r'
009E 346F         00510             retlw   'o'
009F 3463         00511             retlw   'c'
00A0 3468         00512             retlw   'h'
00A1 3469         00513             retlw   'i'
00A2 3470         00514             retlw   'p'
00A3 3420         00515             retlw   ' '
00A4 3454         00516             retlw   'T'
00A5 3465         00517             retlw   'e'
00A6 3463         00518             retlw   'c'
00A7 3468         00519             retlw   'h'
00A8 346E         00520             retlw   'n'
00A9 346F         00521             retlw   'o'
00AA 346C         00522             retlw   'l'
00AB 346F         00523             retlw   'o'
00AC 3467         00524             retlw   'g'
00AD 3479         00525             retlw   'y'
00AE              00526 Table_End
00AE 3400         00527             retlw   0
                  00528 ;
                  00529         if ( (Table & 0x0FF) >= (Table_End & 0x0FF) )
                  00530        MESSG   "Warning - User Definded: Table Table crosses page boundry in computed jump"
                  00531     endif
                  00532 ;
                  00533
                  00534
                  00535
                  00536     end
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)


0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX- ----------------

All other memory blocks unused.

Program Memory Words Used:   172
Program Memory Words Free:  1876
```


© 1997 Microchip Technology Inc.

```
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,    12 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C: 4-BIT DATA INTERFACE, HIGH NIBBLE LISTING

```
MPASM 01.40.01 Intermediate    LM032L.ASM   4-7-1997  9:50:32         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST P=16C64
                     00002         ERRORLEVEL  -302
                     00003 ;
                     00004 ; This program interfaces to a Hitachi (LM032L) 2 line by 20 character display
                     00005 ; module. The program assembles for either 4-bit or 8-bit data interface, depending
                     00006 ; on the value of the 4bit flag. LCD_DATA is the port which supplies the data to
                     00007 ; the LM032L, while LCD_CNTL is the port that has the control lines ( E, RS, RW ).
                     00008 ; In 4-bit mode the data is transfer on the high nibble of the port ( PORT<7:4> ).
                     00009 ;
                     00010 ;       Program = LM032L.ASM
                     00011 ;       Revision Date:   5-10-94
                     00012 ;                        1-22-97    Compatibility with MPASMWIN 1.40
                     00013 ;
                     00014 ;
                     00015        include <p16c64.inc>
                     00001         LIST
                     00002 ; P16C64.INC  Standard Header File, Version 1.01    Microchip Technology, Inc.
                     00238         LIST
                     00016
  0000009F           00017 ADCON1           EQU     9F
                     00018
  00000000           00019 FALSE            EQU     0
  00000001           00020 TRUE             EQU     1
                     00021
                     00022        include <lm032l.h>
                     00069         list
                     00023 ;
  00000000           00024 Four_bit         EQU     FALSE           ; Selects 4- or 8-bit data transfers
  00000001           00025 Data_HI          EQU     TRUE            ; If 4-bit transfers, Hi or Low nibble of PORT
                     00026 ;
                     00027 ;
                     00028     if ( Four_bit && !Data_HI )
                     00029 ;
                     00030 LCD_DATA         EQU     PORTB
                     00031 LCD_DATA_TRIS    EQU     TRISB
```


© 1997 Microchip Technology Inc.

```
                        00032 ;
                        00033     else
                        00034 ;
00000008                00035 LCD_DATA          EQU     PORTD
00000088                00036 LCD_DATA_TRIS     EQU     TRISD
                        00037 ;
                        00038     endif
                        00039 ;
00000005                00040 LCD_CNTL          EQU     PORTA
                        00041 ;
                        00042 ;
                        00043 ;
                        00044 ; LCD Display Commands and Control Signal names.
                        00045 ;
                        00046     if ( Four_bit && !Data_HI )
                        00047 ;
                        00048 E                 EQU     0               ; LCD Enable control line
                        00049 RW                EQU     1               ; LCD Read/Write control line
                        00050 RS                EQU     2               ; LCD Register Select control line
                        00051 ;
                        00052     else
                        00053 ;
00000003                00054 E                 EQU     3               ; LCD Enable control line
00000002                00055 RW                EQU     2               ; LCD Read/Write control line
00000001                00056 RS                EQU     1               ; LCD Register Select control line
                        00057 ;
                        00058     endif
                        00059 ;
                        00060 ;
00000030                00061 TEMP1             EQU     0x030
                        00062 ;
0000                    00063         org       RESET_V                  ; RESET vector location
0000 2808               00064 RESET             GOTO    START            ;
                        00065 ;
                        00066 ; This is the Periperal Interrupt routine. Should NOT get here
                        00067 ;
                        00068     page
0004                    00069         org       ISR_V                    ; Interrupt vector location
0004                    00070 PER_INT_V
0004 1283               00071 ERROR1            BCF     STATUS, RP0      ; Bank 0
0005 1407               00072                   BSF     PORTC, 0
0006 1007               00073                   BCF     PORTC, 0
0007 2804               00074                   GOTO    ERROR1
                        00075 ;
                        00076 ;
                        00077 ;
0008                    00078 START                                      ; POWER_ON Reset (Beginning of program)
```

© 1997 Microchip Technology Inc.

```
0008 0183  00079            CLRF    STATUS              ; Do initialization (Bank 0)
0009 018B  00080            CLRF    INTCON
000A 018C  00081            CLRF    PIR1
000B 1683  00082            BSF     STATUS, RP0         ; Bank 1
000C 3000  00083            MOVLW   0x00                ; The LCD module does not like to work w/ weak pull-ups
000D 0081  00084            MOVWF   OPTION_REG          ;
000E 018C  00085            CLRF    PIE1                ; Disable all peripheral interrupts
           00086 ;***
           00087 ;***   If using device with A/D, these two instructions are required.
           00088 ;***
           00089 ;           MOVLW   0xFF                ;
           00090 ;           MOVWF   ADCON1              ; Port A is Digital.
           00091 ;
           00092 ;
000F 1283  00093            BCF     STATUS, RP0         ; Bank 0
0010 0185  00094            CLRF    PORTA               ; ALL PORT output should output Low.
0011 0186  00095            CLRF    PORTB
0012 0187  00096            CLRF    PORTC
0013 0188  00097            CLRF    PORTD
0014 0189  00098            CLRF    PORTE
0015 1010  00099            BCF     T1CON, TMR1ON       ; Timer 1 is NOT incrementing
           00100 ;
0016 1683  00101            BSF     STATUS, RP0         ; Select Bank 1
0017 0185  00102            CLRF    TRISA               ; RA5 -  0 outputs
0018 30F0  00103            MOVLW   0xF0                ;
0019 0086  00104            MOVWF   TRISB               ; RB7 - 4 inputs, RB3 - 0 outputs
001A 0187  00105            CLRF    TRISC               ; RC Port are outputs
001B 1407  00106            BSF     TRISC, T1OSO        ; RC0 needs to be input for the oscillator to function
001C 0188  00107            CLRF    TRISD               ; RD Port are outputs
001D 0189  00108            CLRF    TRISE               ; RE Port are outputs
001E 140C  00109            BSF     PIE1, TMR1IE        ; Enable TMR1 Interrupt
001F 1781  00110            BSF     OPTION_REG,NOT_RBPU ; Disable PORTB pull-ups
0020 1283  00111            BCF     STATUS, RP0         ; Select Bank 0
           00112 ;
           00113     page
           00114 ;
           00115 ; Initilize the LCD Display Module
           00116 ;
0021 0185  00117            CLRF    LCD_CNTL            ; ALL PORT output should output Low.
           00118
0022       00119 DISPLAY_INIT
           00120     if ( Four_bit && !Data_HI )
           00121             MOVLW   0x02                ; Command for 4-bit interface low nibble
           00122     endif
           00123 ;
           00124     if ( Four_bit && Data_HI )
           00125             MOVLW   0x020               ; Command for 4-bit interface high nibble
```


© 1997 Microchip Technology Inc.

```
                    00126     endif
                    00127 ;
                    00128     if ( !Four_bit )
0022 3038           00129             MOVLW   0x038               ; Command for 8-bit interface
                    00130     endif
                    00131 ;
0023 0088           00132             MOVWF   LCD_DATA            ;
0024 1585           00133             BSF     LCD_CNTL, E         ;
0025 1185           00134             BCF     LCD_CNTL, E         ;
                    00135 ;
                    00136 ; This routine takes the calculated times that the delay loop needs to
                    00137 ; be executed, based on the LCD_INIT_DELAY EQUate that includes the
                    00138 ; frequency of operation. These use registers before they are needed to
                    00139 ; store the time.
                    00140 ;
0026 3006           00141 LCD_DELAY   MOVLW   LCD_INIT_DELAY      ;
0027 00B3           00142             MOVWF   MSD                 ; Use MSD and LSD Registers to Initilize LCD
0028 01B4           00143             CLRF    LSD                 ;
0029 0BB4           00144 LOOP2       DECFSZ  LSD, F              ; Delay time = MSD * ((3 * 256) + 3) * Tcy
002A 2829           00145             GOTO    LOOP2               ;
002B 0BB3           00146             DECFSZ  MSD, F              ;
002C                00147 END_LCD_DELAY
002C 2829           00148             GOTO    LOOP2               ;
                    00149 ;
                    00150 ; Command sequence for 2 lines of 5x7 characters
                    00151 ;
002D                00152 CMD_SEQ
                    00153 ;
                    00154     if ( Four_bit )
                    00155         if ( !Data_HI )
                    00156             MOVLW   0X02                ; 4-bit low nibble xfer
                    00157         else
                    00158             MOVLW   0X020               ; 4-bit high nibble xfer
                    00159         endif
                    00160 ;
                    00161     else                                ; 8-bit mode
002D 3038           00162             MOVLW   0X038
                    00163     endif
                    00164 ;
002E 0088           00165             MOVWF   LCD_DATA            ; This code for both 4-bit and 8-bit modes
002F 1585           00166             BSF     LCD_CNTL, E         ;
0030 1185           00167             BCF     LCD_CNTL, E         ;
                    00168 ;
                    00169     if ( Four_bit )                     ; This code for only 4-bit mode (2nd xfer)
                    00170         if ( !Data_HI )
                    00171             MOVLW   0x08                ; 4-bit low nibble xfer
                    00172         else
```

© 1997 Microchip Technology Inc.

```
                    00173             MOVLW   0x080             ; 4-bit high nibble xfer
                    00174         endif
                    00175             MOVWF   LCD_DATA          ;
                    00176             BSF     LCD_CNTL, E       ;
                    00177             BCF     LCD_CNTL, E       ;
                    00178     endif
                    00179 ;
                    00180 ; Busy Flag should be valid after this point
                    00181 ;
0031 300C           00182             MOVLW   DISP_ON           ;
0032 2068           00183             CALL    SEND_CMD          ;
0033 3001           00184             MOVLW   CLR_DISP          ;
0034 2068           00185             CALL    SEND_CMD          ;
0035 3006           00186             MOVLW   ENTRY_INC         ;
0036 2068           00187             CALL    SEND_CMD          ;
0037 3080           00188             MOVLW   DD_RAM_ADDR       ;
0038 2068           00189             CALL    SEND_CMD          ;
                    00190 ;
                    00191     page
                    00192 ;
                    00193 ;Send a message the hard way
0039 304D           00194             movlw   'M'
003A 205F           00195             call    SEND_CHAR
003B 3069           00196             movlw   'i'
003C 205F           00197             call    SEND_CHAR
003D 3063           00198             movlw   'c'
003E 205F           00199             call    SEND_CHAR
003F 3072           00200             movlw   'r'
0040 205F           00201             call    SEND_CHAR
0041 306F           00202             movlw   'o'
0042 205F           00203             call    SEND_CHAR
0043 3063           00204             movlw   'c'
0044 205F           00205             call    SEND_CHAR
0045 3068           00206             movlw   'h'
0046 205F           00207             call    SEND_CHAR
0047 3069           00208             movlw   'i'
0048 205F           00209             call    SEND_CHAR
0049 3070           00210             movlw   'p'
004A 205F           00211             call    SEND_CHAR
                    00212
004B 30C0           00213             movlw   B'11000000'       ;Address DDRam first character, second line
004C 2068           00214             call    SEND_CMD
                    00215
                    00216                                       ;Demonstration of the use of a table to output a message
004D 3000           00217             movlw   0                 ;Table address of start of message
004E                00218 dispmsg
004E 00B0           00219             movwf   TEMP1             ;TEMP1 holds start of message address
```


© 1997 Microchip Technology Inc.

```
004F 2083          00220            call    Table
0050 39FF          00221            andlw   0FFh              ;Check if at end of message (zero
0051 1903          00222            btfsc   STATUS,Z          ;returned at end)
0052 2857          00223            goto    out
0053 205F          00224            call    SEND_CHAR         ;Display character
0054 0830          00225            movf    TEMP1,w           ;Point to next character
0055 3E01          00226            addlw   1
0056 284E          00227            goto    dispmsg
0057               00228 out
0057               00229 loop
0057 2857          00230            goto    loop              ;Stay here forever
                   00231 ;
                   00232 ;
0058               00233 INIT_DISPLAY
0058 300C          00234            MOVLW   DISP_ON           ; Display On, Cursor On
0059 2068          00235            CALL    SEND_CMD          ; Send This command to the Display Module
005A 3001          00236            MOVLW   CLR_DISP          ; Clear the Display
005B 2068          00237            CALL    SEND_CMD          ; Send This command to the Display Module
005C 3006          00238            MOVLW   ENTRY_INC         ; Set Entry Mode Inc., No shift
005D 2068          00239            CALL    SEND_CMD          ; Send This command to the Display Module
005E 0008          00240            RETURN
                   00241 ;
                   00242     page
                   00243 ;
                   00244 ;*****************************************************************
                   00245 ;* The LCD Module Subroutines                                    *
                   00246 ;*****************************************************************
                   00247 ;
                   00248     if ( Four_bit )      ; 4-bit Data transfers?
                   00249 ;
                   00250         if ( Data_HI )   ; 4-bit transfers on the high nibble of the PORT
                   00251 ;
                   00252 ;*****************************************************************
                   00253 ;*SendChar - Sends character to LCD                            *
                   00254 ;*This routine splits the character into the upper and lower   *
                   00255 ;*nibbles and sends them to the LCD, upper nibble first.       *
                   00256 ;*****************************************************************
                   00257 ;
                   00258 SEND_CHAR
                   00259            MOVWF   CHAR              ;Character to be sent is in W
                   00260            CALL    BUSY_CHECK        ;Wait for LCD to be ready
                   00261            MOVF    CHAR, w
                   00262            ANDLW   0xF0              ;Get upper nibble
                   00263            MOVWF   LCD_DATA          ;Send data to LCD
                   00264            BCF     LCD_CNTL, RW      ;Set LCD to read
                   00265            BSF     LCD_CNTL, RS      ;Set LCD to data mode
                   00266            BSF     LCD_CNTL, E       ;toggle E for LCD
```

© 1997 Microchip Technology Inc.

```
00267             BCF     LCD_CNTL, E
00268             SWAPF   CHAR, w
00269             ANDLW   0xF0              ; Get lower nibble
00270             MOVWF   LCD_DATA          ; Send data to LCD
00271             BSF     LCD_CNTL, E       ; toggle E for LCD
00272             BCF     LCD_CNTL, E
00273             RETURN
00274 ;
00275         else                          ; 4-bit transfers on the low nibble of the PORT
00276 ;
00277 ;*********************************************************************
00278 ;* SEND_CHAR - Sends character to LCD                                 *
00279 ;* This routine splits the character into the upper and lower         *
00280 ;* nibbles and sends them to the LCD, upper nibble first.             *
00281 ;* The data is transmitted on the PORT<3:0> pins                      *
00282 ;*********************************************************************
00283 ;
00284 SEND_CHAR
00285             MOVWF   CHAR              ; Character to be sent is in W
00286             CALL    BUSY_CHECK        ; Wait for LCD to be ready
00287             SWAPF   CHAR, W
00288             ANDLW   0x0F              ; Get upper nibble
00289             MOVWF   LCD_DATA          ; Send data to LCD
00290             BCF     LCD_CNTL, RW      ; Set LCD to read
00291             BSF     LCD_CNTL, RS      ; Set LCD to data mode
00292             BSF     LCD_CNTL, E       ; toggle E for LCD
00293             BCF     LCD_CNTL, E
00294             MOVF    CHAR, W
00295             ANDLW   0x0F              ; Get lower nibble
00296             MOVWF   LCD_DATA          ; Send data to LCD
00297             BSF     LCD_CNTL, E       ; toggle E for LCD
00298             BCF     LCD_CNTL, E
00299             RETURN
00300 ;
00301         endif
00302     else
00303 ;
00304 ;*********************************************************************
00305 ;* SEND_CHAR - Sends character contained in register W to LCD     *
00306 ;* This routine sends the entire character to the PORT            *
00307 ;* The data is transmitted on the PORT<7:0> pins                  *
00308 ;*********************************************************************
00309 ;
005F              00310 SEND_CHAR
005F 00B6         00311             MOVWF   CHAR              ; Character to be sent is in W
0060 2071         00312             CALL    BUSY_CHECK        ; Wait for LCD to be ready
0061 0836         00313             MOVF    CHAR, w
```


© 1997 Microchip Technology Inc.

```
0062 0088           00314             MOVWF   LCD_DATA          ; Send data to LCD
0063 1105           00315             BCF     LCD_CNTL, RW      ; Set LCD in read mode
0064 1485           00316             BSF     LCD_CNTL, RS      ; Set LCD in data mode
0065 1585           00317             BSF     LCD_CNTL, E       ; toggle E for LCD
0066 1185           00318             BCF     LCD_CNTL, E
0067 0008           00319             RETURN
                    00320 ;
                    00321     endif
                    00322 ;
                    00323     page
                    00324 ;
                    00325 ;*****************************************************************
                    00326 ;* SendCmd - Sends command to LCD                               *
                    00327 ;* This routine splits the command into the upper and lower     *
                    00328 ;* nibbles and sends them to the LCD, upper nibble first.       *
                    00329 ;* The data is transmitted on the PORT<3:0> pins                *
                    00330 ;*****************************************************************
                    00331 ;
                    00332     if ( Four_bit )                 ; 4-bit Data transfers?
                    00333 ;
                    00334         if ( Data_HI )              ; 4-bit transfers on the high nibble of the PORT
                    00335 ;
                    00336 ;*****************************************************************
                    00337 ;* SEND_CMD - Sends command to LCD                              *
                    00338 ;* This routine splits the command into the upper and lower     *
                    00339 ;* nibbles and sends them to the LCD, upper nibble first.       *
                    00340 ;*****************************************************************
                    00341
                    00342 SEND_CMD
                    00343             MOVWF   CHAR              ; Character to be sent is in W
                    00344             CALL    BUSY_CHECK        ; Wait for LCD to be ready
                    00345             MOVF    CHAR,w
                    00346             ANDLW   0xF0              ; Get upper nibble
                    00347             MOVWF   LCD_DATA          ; Send data to LCD
                    00348             BCF     LCD_CNTL,RW       ; Set LCD to read
                    00349             BCF     LCD_CNTL,RS       ; Set LCD to command mode
                    00350             BSF     LCD_CNTL,E        ; toggle E for LCD
                    00351             BCF     LCD_CNTL,E
                    00352             SWAPF   CHAR,w
                    00353             ANDLW   0xF0              ; Get lower nibble
                    00354             MOVWF   LCD_DATA          ; Send data to LCD
                    00355             BSF     LCD_CNTL,E        ; toggle E for LCD
                    00356             BCF     LCD_CNTL,E
                    00357             RETURN
                    00358 ;
                    00359         else                          ; 4-bit transfers on the low nibble of the PORT
                    00360 ;
```

© 1997 Microchip Technology Inc.

```
                00361 SEND_CMD
                00362             MOVWF   CHAR              ; Character to be sent is in W
                00363             CALL    BUSY_CHECK        ; Wait for LCD to be ready
                00364             SWAPF   CHAR, W
                00365             ANDLW   0x0F              ; Get upper nibble
                00366             MOVWF   LCD_DATA          ; Send data to LCD
                00367             BCF     LCD_CNTL, RW      ; Set LCD to read
                00368             BCF     LCD_CNTL, RS      ; Set LCD to command mode
                00369             BSF     LCD_CNTL, E       ; toggle E for LCD
                00370             BCF     LCD_CNTL, E
                00371             MOVF    CHAR, W
                00372             ANDLW   0x0F              ; Get lower nibble
                00373             MOVWF   LCD_DATA          ; Send data to LCD
                00374             BSF     LCD_CNTL, E       ; toggle E for LCD
                00375             BCF     LCD_CNTL, E
                00376             RETURN
                00377 ;
                00378         endif
                00379     else
                00380 ;
                00381 ;*****************************************************************
                00382 ;* SEND_CND - Sends command contained in register W to LCD     *
                00383 ;* This routine sends the entire character to the PORT         *
                00384 ;* The data is transmitted on the PORT<7:0> pins               *
                00385 ;*****************************************************************
                00386
0068            00387 SEND_CMD
0068 00B6       00388             MOVWF   CHAR              ; Command to be sent is in W
0069 2071       00389             CALL    BUSY_CHECK        ; Wait for LCD to be ready
006A 0836       00390             MOVF    CHAR, w
006B 0088       00391             MOVWF   LCD_DATA          ; Send data to LCD
006C 1105       00392             BCF     LCD_CNTL, RW      ; Set LCD in read mode
006D 1085       00393             BCF     LCD_CNTL, RS      ; Set LCD in command mode
006E 1585       00394             BSF     LCD_CNTL, E       ; toggle E for LCD
006F 1185       00395             BCF     LCD_CNTL, E
0070 0008       00396             RETURN
                00397 ;
                00398     endif
                00399 ;
                00400     page
                00401 ;
                00402     if ( Four_bit )                   ; 4-bit Data transfers?
                00403 ;
                00404         if ( Data_HI )                ; 4-bit transfers on the high nibble of the PORT
                00405 ;
                00406 ;*****************************************************************
                00407 ;* This routine checks the busy flag, returns when not busy      *
```


© 1997 Microchip Technology Inc.

```
00408 ;*  Affects:                                                          *
00409 ;*       TEMP - Returned with busy/address                           *
00410 ;**********************************************************************
00411 ;
00412 BUSY_CHECK
00413             BSF     STATUS, RP0         ; Select Register Bank1
00414             MOVLW   0xFF                ; Set Port_D for input
00415             MOVWF   LCD_DATA_TRIS
00416             BCF     STATUS, RP0         ; Select Register Bank0
00417             BCF     LCD_CNTL, RS        ; Set LCD for Command mode
00418             BSF     LCD_CNTL, RW        ; Setup to read busy flag
00419             BSF     LCD_CNTL, E         ; Set E high
00420             BCF     LCD_CNTL, E         ; Set E low
00421             MOVF    LCD_DATA, W         ; Read upper nibble busy flag, DDRam address
00422             ANDLW   0xF0                ; Mask out lower nibble
00423             MOVWF   TEMP
00424             BSF     LCD_CNTL, E         ; Toggle E to get lower nibble
00425             BCF     LCD_CNTL, E
00426             SWAPF   LCD_DATA, w         ; Read lower nibble busy flag, DDRam address
00427             ANDLW   0x0F                ; Mask out upper nibble
00428             IORWF   TEMP, F             ; Combine nibbles
00429             BTFSC   TEMP, 7             ; Check busy flag, high = busy
00430             GOTO    BUSY_CHECK          ; If busy, check again
00431             BCF     LCD_CNTL, RW
00432             BSF     STATUS, RP0         ; Select Register Bank1
00433             MOVLW   0x0F
00434             MOVWF   LCD_DATA_TRIS       ; Set Port_D for output
00435             BCF     STATUS, RP0         ; Select Register Bank0
00436             RETURN
00437 ;
00438         else                            ; 4-bit transfers on the low nibble of the PORT
00439 ;
00440 ;**********************************************************************
00441 ;* This routine checks the busy flag, returns when not busy           *
00442 ;*  Affects:                                                          *
00443 ;*       TEMP - Returned with busy/address                           *
00444 ;**********************************************************************
00445 ;
00446 BUSY_CHECK
00447             BSF     STATUS, RP0         ; Bank 1
00448             MOVLW   0xFF                ; Set PortB for input
00449             MOVWF   LCD_DATA_TRIS
00450             BCF     STATUS, RP0         ; Bank 0
00451             BCF     LCD_CNTL, RS        ; Set LCD for Command mode
00452             BSF     LCD_CNTL, RW        ; Setup to read busy flag
00453             BSF     LCD_CNTL, E         ; Set E high
00454             BCF     LCD_CNTL, E         ; Set E low
```

© 1997 Microchip Technology Inc.

```
                00455             SWAPF   LCD_DATA, W       ; Read upper nibble busy flag, DDRam address
                00456             ANDLW   0xF0              ; Mask out lower nibble
                00457             MOVWF   TEMP              ;
                00458             BSF     LCD_CNTL, E       ; Toggle E to get lower nibble
                00459             BCF     LCD_CNTL, E
                00460             MOVF    LCD_DATA, W       ; Read lower nibble busy flag, DDRam address
                00461             ANDLW   0x0F              ; Mask out upper nibble
                00462             IORWF   TEMP, F           ; Combine nibbles
                00463             BTFSC   TEMP, 7           ; Check busy flag, high = busy
                00464             GOTO    BUSY_CHECK        ; If busy, check again
                00465             BCF     LCD_CNTL, RW
                00466             BSF     STATUS, RP0       ; Bank 1
                00467             MOVLW   0xF0              ;
                00468             MOVWF   LCD_DATA_TRIS     ; RB7 - 4 = inputs, RB3 - 0 = output
                00469             BCF     STATUS, RP0       ; Bank 0
                00470             RETURN
                00471 ;
                00472         endif
                00473     else
                00474 ;
                00475 ;*****************************************************************
                00476 ;* This routine checks the busy flag, returns when not busy    *
                00477 ;*  Affects:                                                     *
                00478 ;*       TEMP - Returned with busy/address                      *
                00479 ;*****************************************************************
                00480 ;
0071            00481 BUSY_CHECK
0071 1683       00482             BSF     STATUS,RP0        ; Select Register Bank1
0072 30FF       00483             MOVLW   0xFF              ; Set port_D for input
0073 0088       00484             MOVWF   LCD_DATA_TRIS
0074 1283       00485             BCF     STATUS, RP0       ; Select Register Bank0
0075 1085       00486             BCF     LCD_CNTL, RS      ; Set LCD for command mode
0076 1505       00487             BSF     LCD_CNTL, RW      ; Setup to read busy flag
0077 1585       00488             BSF     LCD_CNTL, E       ; Set E high
0078 1185       00489             BCF     LCD_CNTL, E       ; Set E low
0079 0808       00490             MOVF    LCD_DATA, w       ; Read busy flag, DDram address
007A 00B5       00491             MOVWF   TEMP
007B 1BB5       00492             BTFSC   TEMP, 7           ; Check busy flag, high=busy
007C 2871       00493             GOTO    BUSY_CHECK
007D 1105       00494             BCF     LCD_CNTL, RW
007E 1683       00495             BSF     STATUS, RP0       ; Select Register Bank1
007F 3000       00496             MOVLW   0x00
0080 0088       00497             MOVWF   LCD_DATA_TRIS     ; Set port_D for output
0081 1283       00498             BCF     STATUS, RP0       ; Select Register Bank0
0082 0008       00499             RETURN
                00500 ;
                00501     endif
```


© 1997 Microchip Technology Inc.

```
                        00502     page
                        00503 ;
0083                    00504 Table
0083 0782               00505             addwf   PCL, F              ; Jump to char pointed to in W reg
0084 344D               00506             retlw   'M'
0085 3469               00507             retlw   'i'
0086 3463               00508             retlw   'c'
0087 3472               00509             retlw   'r'
0088 346F               00510             retlw   'o'
0089 3463               00511             retlw   'c'
008A 3468               00512             retlw   'h'
008B 3469               00513             retlw   'i'
008C 3470               00514             retlw   'p'
008D 3420               00515             retlw   ' '
008E 3454               00516             retlw   'T'
008F 3465               00517             retlw   'e'
0090 3463               00518             retlw   'c'
0091 3468               00519             retlw   'h'
0092 346E               00520             retlw   'n'
0093 346F               00521             retlw   'o'
0094 346C               00522             retlw   'l'
0095 346F               00523             retlw   'o'
0096 3467               00524             retlw   'g'
0097 3479               00525             retlw   'y'
0098                    00526 Table_End
0098 3400               00527             retlw   0
                        00528 ;
                        00529         if ( (Table & 0x0FF) >= (Table_End & 0x0FF) )
                        00530        MESSG   "Warning - User Definded: Table Table crosses page boundry in computed jump"
                        00531     endif
                        00532 ;
                        00533
                        00534
                        00535
                        00536     end
```


© 1997 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXX------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   150
Program Memory Words Free:  1898


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,    12 suppressed
```


© 1997 Microchip Technology Inc.

A PC-Based Development Programmer for the PIC16C84

Author: *Robert Spur*
Analog Design Specialist, Inc.

INTRODUCTION

This application note describes the construction of a low cost serial programmer which uses a PC with a parallel (Centronix printer) port to control a PIC16C84. This programmer has the capability of programming a PIC16C84 microcontroller, and reading back internal data without removing the device from the target circuit.

This feature is very useful in applications where changes in program code or program constants are necessary to compensate for other system features. For example, an embedded control system may have to compensate for variances in a mechanical actuator's performance or loading. The basic program can be programmed and tested during design phase. The final program and control constants can be easily added later in the production phase without removing the microcontroller from the circuit.

Automatic software and performance upgrades can also be implemented in-system. Upon receiving new system software via disk or modem, a control processor with the included programming code could perform in-circuit reprogramming of other microcontrollers in the system.

This programmer can load program code, part configuration, and EEPROM data into the PIC16C84. In read back mode, it can verify all data entries.

PROGRAMMING DESCRIPTION

The PIC16C84 microcontroller is placed into programming mode by forcing a low logic level on RB7 (pin 13) and RB6 (pin 12) while $\overline{MCLR}$ (pin 4) is first brought low to reset the part, and then brought to the program/verify voltage of 12 to 14V. The $\overline{MCLR}$ pin remains at the program/verify voltage for the remainder of the programming or verification.

After entering programming mode, RB7 is used to serially enter programming modes and data into the part. A high to low transition on RB6, the clock input, qualifies each bit of the data applied on RB7. Please refer to the PIC16C84 Programming Specification (DS30189) for details on the figures. The serial command-data format is specified in Figure 1.2.1.3 of the Microchip PIC16C84 Programming Specification (DS30189). The first 6 bits form the command field, and the last 16 bits form the data field. Notice that the data field is composed of one zero starting bit, 14 actual data bits, and one zero stop bit. The increment address command, shown in Figure 1.2.1.5 (PIC16C84 Programming Specification, DS30189), is comprised of only the command field. Table 1.2.1.1 (see DS30189) summarizes the available commands and command codes for serial programming mode.

Read mode is similar to programming mode with the exception that the data direction of RB7 is reversed after receiving the 6-bit command to allow the requested data to be returned to the programmer. Figure 1.2.1.4 (see DS30189) shows this sequence which starts by shifting the 6-bit command into the part. After the read command is issued, the programmer tri-states its buffer to allow the part to serially shift its internal data back to the programmer. The rising edge of RB6, (the clock input), controls the data flow by sequentially shifting previously programmed or data bits from the part. The programmer qualifies this data on the falling edge of RB6. Notice that 16 clock cycles are necessary to shift out 14 data bits.

Accidental in-circuit reprogramming is prevented during normal operation by the $\overline{MCLR}$ voltage which should never exceed the maximum circuit supply voltage of 6 VDC and the logic levels of port bits RB7 and RB6.

After program/verification the $\overline{MCLR}$ pin is brought low to reset the target microcontroller and then electrically released. The target circuit is then free to activate the $\overline{MCLR}$ signal. In the event $\overline{MCLR}$ is not forced by the target circuit, R4 (a 2 kΩ pull-up resistor in the programmer) provides a high logic level on the target microcontroller which enables execution of its program independent of the programmer connection. Provisions should be made to prevent the target circuit from resetting the target microcontroller with $\overline{MCLR}$ or affecting the RB7 and RB6 pins during the programming process. In most cases this can be done without jumpers.

© 1997 Microchip Technology Inc.

DETAILED CIRCUIT DESCRIPTION

A logic high on PC parallel interface latch bit D4 turns on Q3 causing the $\overline{MCLR}$ pin to go low which places the target part in reset mode. The reset condition is then removed and the program/verify voltage is applied by placing a logic high on D3 and a logic low on D4 which turns off Q3 and turns on Q2 and Q1. Circuit protection of Q1 and Q3 is obtained from connecting the emitter of Q2 to latch bit D4 which prevents a simultaneous reset and program/verify voltage mode. Q2, a 2N3904, has a reverse emitter base breakdown voltage of 6V which will not be exceeded when 5V logic is used on the parallel interface.

Resistors R1, R2, R3, and diode D1 provide a logic level interface to the analog circuitry. R4 provides a $\overline{MCLR}$ (master clear) pull-up function during target circuit run mode. The programming voltage is supplied and adjusted by an external lab supply. This supply should have a current limit in the 100 mA range. 5V for U2 (LS244) is locally regulated from programming supply voltage by U1. R5 (750Ω resistor) is connected to the regulator output to insure proper 5V regulation when the 13.5V programming voltage is applied through the pull-up resistor R4.

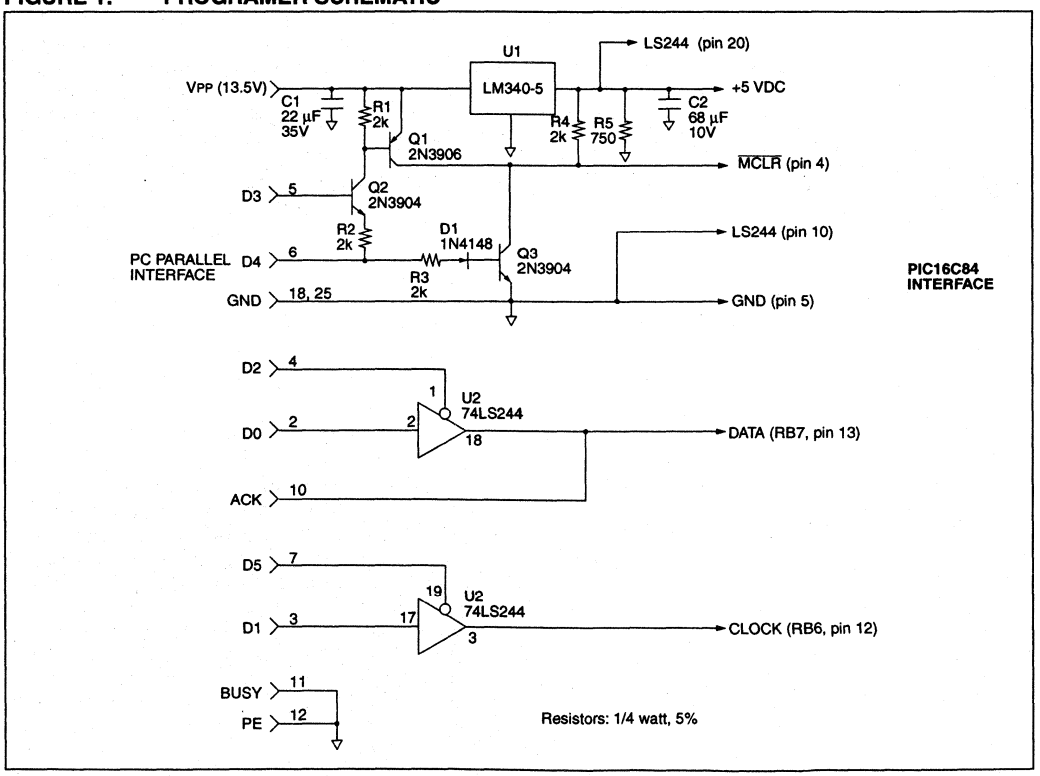
Data and clock are connected to the part via tri-state buffer, U2. PC parallel port interface bit D0 is used for data and port bit D1 is used for clock. During programming mode both clock and data buffers are enabled by port bits D2 and D5. During read mode, the data buffer is tri-stated via D2 and the printer data acknowledge signal line is used to receive verification data from the part.

After program/verification mode both the data and clock lines are tri-stated via D2 and D5, allowing these lines to be used by the target circuit. This allows the programmer to remain physically, but not electrically connected to the target system.

An optional 5V line was included in the 3-foot programming interconnect cable for convenience. Short interconnection leads and good grounding are always good construction practice.

To meet the programming/verification specification, the target part's supply voltage should first be set to the maximum specified supply voltage and a program/data read back should then be performed. This process is then repeated at the lowest specified supply voltage.

FIGURE 1: PROGRAMER SCHEMATIC



© 1997 Microchip Technology Inc.

SOFTWARE DESCRIPTION

The listed code provides a hardware-software interface to a standard PC parallel (Centronix) interface port. The code can be adapted to a microprocessor parallel interface port by substituting an output command for the "biosprint" command.

Control software can transfer the PIC16C84 program, configuration bits, and EEPROM data from a standard PROM interface file into the target system by reading the file and calling the function in Example 1 using the appropriate command name in the definition table, and the data to be programmed. The command names are repeated here for reference.

LOAD_CONFIG	Sets PIC16C84 data pointer to configuration.
LOAD_DATA	Loads, but does not program, data.
READ_DATA	Reads data at current pointer location.
INC_ADDR	Increments PIC16C84 data pointer.
BEGIN_PROG	Programs data at current data pointer location.
PARALLEL_MODE	Puts PIC16C84 into parallel mode (not used).
LOAD_DATA_DM	Loads EEPROM data.
READ_DATA_DM	Reads EEPROM data.

Function "int ser_pic16c84(<command>,<data [or 0]>) is called to perform command. Function returns internal data after read commands.

Do not forget to initiate the programming mode before programming, increment the addresses after each byte is programmed, and put the programmer in run mode after programming.

Designed by: Analog Design Specialist, Inc.
P.O. Box 26-0846
Littleton, CO 80126

3

EXAMPLE 1: PUT TARGET SYSTEM INTO PROGRAM MODE

```
.. program code..
ser_pic16c84(PROGRAM_MODE,0);
.. program code..
```

EXAMPLE 2: READ DATA FROM THE TARGET SYSTEM

```
.. program code..
data = ser_pic16c84(READ_DATA,0); // read data
// transfers data from target part to variable "data"
.. more program code..
```

EXAMPLE 3: PROGRAM DATA INTO THE TARGET SYSTEM

```
.. program code..
ser_pic16c84(LOAD_DATA,data);// load data into target
ser_pic16c84(BEGIN_PROG,0);// program loaded data
ser_pic16c84(INC_ADDR,0);// increment to next address
 // transfers data from program variable "data" to target part
.. more program code..
```

EXAMPLE 4: PUT TARGET SYSTEM INTO RUN MODE

```
.. program code..
ser_pic16c84(RUN,0);
.. program code..
```

© 1997 Microchip Technology Inc.

```
//************************** FIGURE #2 ********************************
//**                                                                 **
//**   SERIAL PROGRAMMING ROUTINE FOR THE PIC16C84 MICROCONTROLLER   **
//**                                                                 **
//**                    Analog Design Specialists                    **
//**                                                                 **
//*********************************************************************

//FUNCTION PROTOTYPE: int ser_pic16c84(int cmd, int data)

// cmd: LOAD_CONFIG   -> part configuration bits
//      LOAD_DATA     -> program data, write
//      READ_DATA     -> program data, read
//      INC_ADDR      -> increment to the next address (routine does not auto increment)
//      BEGIN_PROG    -> program a previously loaded program code or data
//      LOAD_DATA_DM  -> load EEPROM data regesters (BEGIN_PROG must follow)
//      READ_DATA_DM  -> read EEPROM data
//
// data: 1) 14 bits of program data or
//       2)  8 bits of EEPROM data (least significant 8 bits of int)

// Additional programmer commands (not part of PIC16C84 programming codes)
//
// cmd: RESET         -> provides 1 ms reset pulse to target system
//      PROGRAM_MODE  -> initializes PIC16C84 for programming
//      RUN           -> disconnects programmer from target system
//
// function returns:1) 14 or 8 bits read back data for read commands
//                  2) zero                        for write data commands
//                  3) PIC_PROG_EROR = -1 for programming function errors
//                  4) PROGMR_ERROR  = -2 for programmer function errors


#include <bios.h>

#define LOAD_CONFIG    0
#define LOAD_DATA      2
#define READ_DATA      4
#define INC_ADDR       6
#define BEGIN_PROG     8
#define PARALLEL_MODE 10  // not used
#define LOAD_DATA_DM   3
#define READ_DATA_DM   5
#define MAX_PIC_CMD   63  // division between pic16c84 and programmer commands

#define RESET         64  // external reset command, not needed for programming
#define PROGRAM_MODE  65  // initialize program mode
#define RUN           66  // electrically disconnect programmer

#define PIC_PROG_EROR -1
#define PROGMR_ERROR  -2

#define PTR            0  // use device #0

// parallel port bits
//     d0: data output to part to be programmed
//     d1: programming clock
//     d2: data dirrection, 0= enable tri state buf -> send data to part
//     d3: Vpp control 1= turn on Vpp
//     d4: ~MCLR =0, 1 = reset device with MCLR line
//     d5: clock line tri state control, 0 = enable clock line

int ser_pic16c84(int cmd, int data)          // custom interface for pic16c84
  {
  int i, s_cmd;
```


© 1997 Microchip Technology Inc.

```
if(cmd <=MAX_PIC_CMD)                          // all programming modes
  {
  biosprint(0,8,PTR);                          // set bits 001000, output mode, clock & data low
  s_cmd = cmd;                                 // retain command "cmd"
  for (i=0;i<6;i++)                            // output 6 bits of command
    {
    biosprint(0,(s_cmd&0x1) +2+8,PTR);         // set bits 001010, clock hi
    biosprint(0,(s_cmd&0x1)   +8,PTR);         // set bits 001000, clock low
    s_cmd >>=1;
    }

  if((cmd ==INC_ADDR)||(cmd ==PARALLEL_MODE)   // command only, no data cycle
    return 0;

  else if(cmd ==BEGIN_PROG)                    // program command only, no data cycle
    {
    delay(10);                                 // 10 ms PIC programming time
    return 0;
    }

  else if((cmd ==LOAD_DATA)||(cmd ==LOAD_DATA_DM)||(cmd ==LOAD_CONFIG)) // output 14 bits
    for (i=200;i;i--) ;                        // delay between command & data
    biosprint(0,2+8,PTR);                      // set bits 001010, clock hi; leading bit
    biosprint(0,  8,PTR);                      // set bits 001000, clock low

    for (i=0;i<14;i++)                         // 14 data bits, lsb first
   {
   biosprint(0,(data&0x1) +2+8,PTR);           // set bits 001010,  clock hi
   biosprint(0,(data&0x1)   +8,PTR);           // set bits 001000,  clock low
   data >>=1;
   }
    biosprint(0,2+8,PTR);                      // set bits 001010, clock hi; trailing bit

// *****************  Analog Design Specialists  ******************

    biosprint(0,  8,PTR);                      // set bits 001000, clock low

    return 0;
    }

  else if((cmd ==READ_DATA)||(cmd ==READ_DATA_DM)) //read 14 bits from part, lsb first
    {
    biosprint(0,  4+8,PTR);                    // set bits 001100, clock low, tri state data buffer
    for (i=200;i;i--) ;                        // delay between command & data
    biosprint(0,2+4+8,PTR);                    // set bits 001110, clock hi, leading bit
    biosprint(0,  4+8,PTR);                    // set bits 001100, clock low

    data =0;
    for (i=0;i<14;i++)                         // input 14 bits of data, lsb first
   {
   data >>=1;                                  // shift data for next input bit
   biosprint(0,2+4+8,PTR);                     // set bits 001110,  clock hi
   biosprint(0,  4+8,PTR);                     // set bits 001100,  clock low
   if(!(biosprint(2,0,0)&0x40)) data += 0x2000; //use printer acknowledge line for input,
                                               //data lsb first
   }
    biosprint(0,2+4+8,PTR);                    // set bits 001110, clock hi, trailing bit
    biosprint(0,  4+8,PTR);                    // set bits 001100, clock low
    return data;
    }

  else return PIC_PROG_EROR;                   // programmer error

  }
else if(cmd == RESET)                          // reset device
```

```
    {
    biosprint(0,32+16+4,PTR);                  // set bits 110100, MCLR = low
(reset                                         // PIC16C84), programmer not connected
    delay(1);                                  // 1ms delay
    biosprint(0,32   +4,PTR);                  // set bits 100100, MCLR = high
    return 0;
    }

  else if(cmd ==PROGRAM_MODE)                  // enter program mode
    {
    biosprint(0,32+16+4,PTR);                  // set bits 110100, Vpp off, MCLR =low
                                               //(reset PIC16C84)
    delay(10);                                 //10 ms, allow programming voltage to stabilize

    biosprint(0,8,PTR);                        // set bits 001000, Vpp on , MCLR = 13.5 volts,
                                               // clock & data connected
    delay(10);                                 // 10 ms, allow programming voltage to stabilize

    return 0;
    }

  else if(cmd ==RUN)                           // disconnects programmer from device
    {
    biosprint(0,32+4,PTR);                     // set bits 100100
    return 0;
    }
  else return PROGMR_ERROR;                    // command error
  }
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN591

Apple® Desktop Bus (ADB™)

Author: *Rob McCall*
WFT Electronics
Support: *Gus Calabrese*
Dave Evink
Curt Apperson
WFT Electronics

INTRODUCTION

The purpose of this application note is to introduce a PIC16CXXX based ADB interface which can be used as a basis for the development of custom ADB devices. This application note describes; the hardware involved, a general purpose ADB protocol handler, and an example application task. The example software application supports a single key keyboard to the Macintosh® computer (Figure 1).

OVERVIEW

ADB licensing from Apple Computer.

Described as a peripheral bus used on almost all Macintoshes (except for the Macintosh 128, 512K, and Plus) for keyboards, mice, etc.

Communication between the ADB task and the application task takes place using several flags. The flags indicate whether there is data received that needs to be sent to the Macintosh, or if data from the Macintosh needs to be sent by the application.

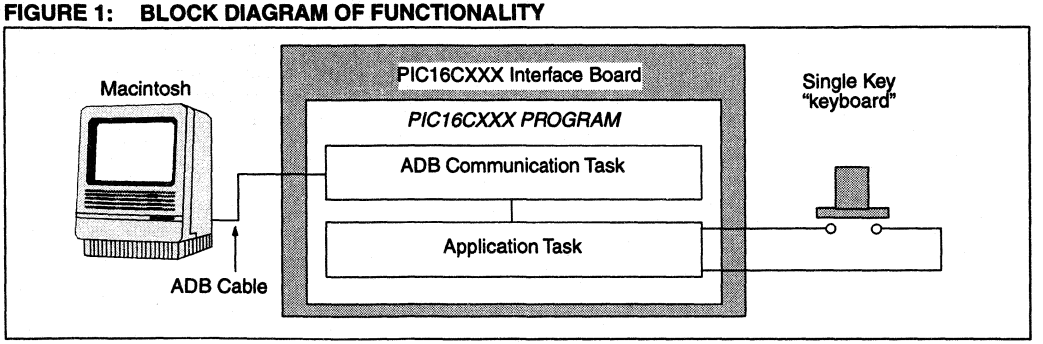
EXPLANATION OF ADB TECHNOLOGY

ADB is an asynchronous pulse-width communication protocol supporting a limited number of devices. All devices share a single I/O wire in a multi-drop master/slave configuration in which any slave device may request service. This is accomplished through a wired OR negative logic arrangement.

The ADB cable is composed of four wires: +5V, gnd, ADB signal, and power-on (of the Macintosh). The signal wire communicates ADB input and output using an open collector type signal. The number of devices is limited by the addressing scheme and a maximum current draw of 500 mA.

Every ADB device has a default address at start-up assigned by Apple. If there are device address conflicts, the protocol supports the reassignment of device addresses at start-up. The software in the PIC16CXXX discussed here is designed to easily modify the device address to make the PICmicro™ appear as another ADB device for testing and development.

FIGURE 1: BLOCK DIAGRAM OF FUNCTIONALITY



Macintosh, and ADB™ (Apple Desktop Bus) are trademarks/registered trademarks of Apple Computer, Inc.

© 1997 Microchip Technology Inc.

No device issues commands, except the host. However, devices are permitted to request service during specific time intervals in the signal/Command protocol. A Service Request is referred to as an "Srq" The signal protocol communication is accomplished by pulling the ADB line low for various time intervals.

The host controls the flow of data through issuance of specific signal sequences and by issuing several types of Commands. The basic command types are Talk, Listen, Flush, and Reserved. Each command has a component called a "Register" indicator which specifies the storage area affected by the command type. The following is a summary explanation of the each of the commands. The complete specifications are available from Apple, as listed in the Resources section of this application note.

PROTOCOL ASSUMPTIONS

The ADB protocol is defined with a number of general assumptions about its use. These assumptions have driven the general philosophy of the communication sequences. It is assumed that the devices on the ADB are used for human input and each are used one at a time, such as a keyboard and a mouse. It is also assumed that the user's transfer time from one device to another is relatively slow. This does not mean that the protocol is limited to these assumptions but rather that the protocol is optimized towards this type of use. This is made very evident in the host polling logic, where the host continues to poll the last device communicated with until another device issues an Srq. Consequently, if another device issues an Srq, the device being communicated with (or the host) may need to retransmit.

ADB Elements:

The ADB protocol has two components, a Signal protocol and a Command/Data protocol. These two elements are intertwined. The Signal protocol is differentiated in most cases by timing periods during which the ADB signal is low. The Apple ADB specification allows ± 3% tolerance timing of the signals from the host and ± 30% by the devices. The signals are:

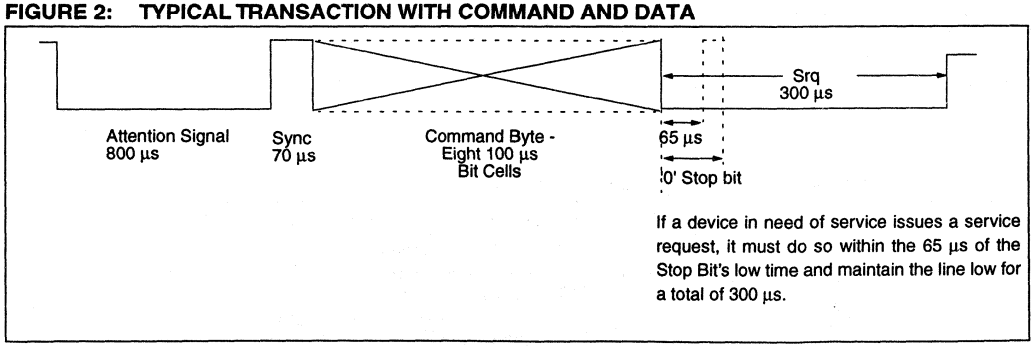
- Reset: signal low for 3 ms.
- Attention: signal low for 800 µs.
- Sync: signal high for 70 µs.
- Stop-to-Start-Time (`Tlt`): signal high for between 65 and 160 µs.
- Service Request (`Srq`): signal low for 300 µs.

After device initialization, in general, all communication through the ADB is accomplished through the following event sequence initiated by the host:

1. Attention signal
2. Sync signal
3. command packet
4. Tlt signal
5. data packet transfer

Depending upon the command, the device may or may not respond with a data packet. Service requests are issued by the devices during a very specific time at the end of the reception of the command packet.

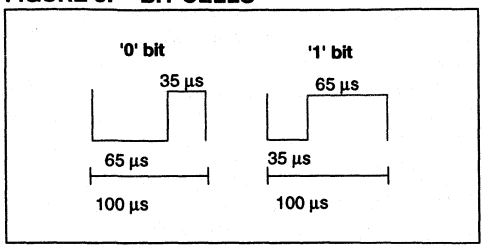
FIGURE 2: TYPICAL TRANSACTION WITH COMMAND AND DATA



© 1997 Microchip Technology Inc.

The command packets and the data packets are the constructs used to communicate the digital information. The method of representing data bits is accomplished in a signal timing construct called a **bit cell**. Each **bit cell** is a 100 µs period. Data '1's and '0's are defined by the proportions of the bit cell time period when the line is low and then high. A '1' bit is represented by the line low for 35 µs, and high for 65 µs. Conversely, A '0' bit is represented by the line low for 65 µs, and high for 35 µs (Figure 3).

FIGURE 3: BIT CELLS



The **Command Packet**, received from the host, follows an Attention signal and a Sync signal. It consists of an 8-bit command byte and a '0' command stop bit. The command byte may be broken down into two nibbles. The upper nibble is a 4-bit unique device address. The lower nibble is defined as a Global or Reserved command for all devices, or a Talk, Listen, or Flush Command for a specific device. Also contained in the lower nibble is a "Register" designator which further details the Command. The importance of the Command Stop Bit Cell is that Srqs' can only be issued by a device to the host during the Command Stop Bit Cell low time if the device address is not for the device wishing service. The Host controls when Srq's are allowed through the Command protocol. The Tlt signal and Data Packet transfer, which are part of every Command packet signal sequence, are overridden if an Srq is issued by any device.

A **Data Packet** is the data sent to, or received from, the host. Its length is variable from 2 to 8 bytes. The structure is a '1' start bit, followed by 2 to 8 bytes, ending with a '0' stop bit. The Apple ADB documentation refers to the data packet sent or requested as Device Data "Registers". This does not necessarily indicate a specific place in memory. In this PIC16CXXX implementation, each Data Register has been limited to two PIC16CXXX register bytes. The ADB specification allows each Data Register to hold between two and eight bytes. They are referenced in the Command byte as "register" 0, 1, 2, or 3. Data Register 3 has special significance. It holds the special status information bits (such as whether Srq's are allowed), the Device Address, and the Device Handler ID. Commands are further defined by the "register id" sent in the Command data packet.

For example, if the Host issues the Command in binary of `0010 1100`, it would be interpreted as "Device 2, Talk Register 0". The complete definition of the Commands and data registers are described in detail in the ADB specifications supplied by Apple.

PIC16CXXX ADB PROTOCOL PROGRAM EVENT SEQUENCE

Overview

At power-on the host will generate a Reset signal. The purpose of Reset is to initialize the devices on the ADB line. This includes determining the addresses of each device, and resolving device address conflicts if there are any. Once the device addresses are determined, each device waits to be commanded or issues an Srq if it requires service from the host and is not being addressed by the host. After Reset processing, the ADB Protocol Task monitors the ADB line for the Attention/Sync/Command signal sequence. The PIC16CXXX program differentiates the signal timing.

Note: The signal detection routines check to see if the Application Task needs service after each event and after the falling edge of the Attention signal is detected.

Command interpretation is accomplished during the low signal time of the Stop Bit cell of the Command packet. Response to the Command must occur after the minimum time of the Stop to Start time period (Tlt), which is 160 µs. but before the max Tlt time of 240 µs. When a device has issued an Srq, it waits to be addressed by the host. If the next Command received is not for that device, it issues the Srq again. The normal response to an Srq will be a Talk Command from the host.

Detailed Description

Start-up

Upon start-up, the Reset routine is executed, looking for the ADB line to be high. When the line is high, an initialization routine is executed during which registers are cleared or loaded with default values. The only exception is a register for generating a random address used in the address conflict resolution process.

Reset

During a Reset condition, default values are loaded, such as the Default Device Address and Handler ID (a piece of information used by the host to identify the type of device). If more than one device has the same address, there is a sequence of events to resolve address conflicts described in the Implementation section. The host assigns a unique address to each device. The Reset condition only takes place once, during start-up, except under unusual conditions, such as testing this program.

© 1997 Microchip Technology Inc.

Attention Routine

When the Reset routine is complete, the Attention Signal routine is executed, looking for the line to go low and then high. This low time is monitored to be within range of the Attention Signal Timing. If the timing is below the minimum threshold, the routine aborts to start over again looking for the line to go low at the beginning of the Attention Signal. If the low time is exceeded, the routine aborts to the Reset Signal routine.

Sync Signal Routine

When the line transitions to high, the Sync Signal routine looks for the line to go low at the start of the first bit of the Command Byte. If the Sync high time is exceeded, the routine aborts to the Attention Signal.

Command Routine

The Command routine detects and decodes the next 8 bit-cells as the Command Byte. The routine must first determine if the device address given is for itself. If the routine determines that the device address in the Command matches the stored device addresses, then it may do one of two things; issue an Srq to the host by holding the line low, or go on to check if the Command is Global to all devices. If the command is Global, the routine determines the specific Command and executes the routine for that Global Command. After execution of the Command routine it then goes back to look for the Attention Signal.

When a device is addressed, it determines whether the Command is to Talk, Listen, or Flush data, for the specified Data Register number. If the Command is for Data Register 3, there are special considerations described for this program in the Implementation section later in this application note. If the Command is to Flush, the routine clears the data in the specified register. The ADB specification defines the action of the Flush Command to be device specific. For a Talk Command or Listen Command, the device then waits for the Tlt signal. When the Command is to Talk, the device sends the data bytes from the specified register and a Data Stop Bit after the Tlt minimum time. For a Listen Command, the device receives data for the specified register.

When the data has been Flushed, Sent, or Received, the device then returns to monitoring for the Attention signal again.

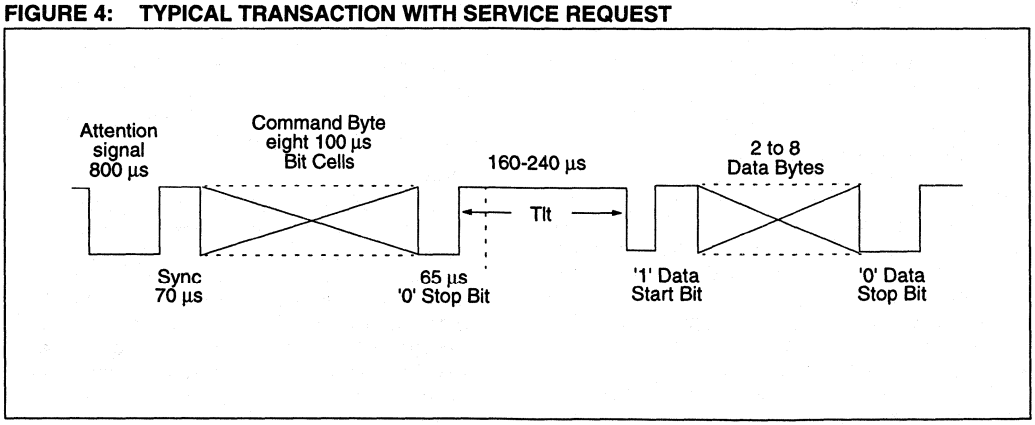
Note 1: In this PIC16CXXX program, the Application Task is serviced before looking for the Attention signal.

Note 2: If at any time the line is low or high outside of the timing ranges, the program aborts to check if an Attention or Reset signal has been issued by the Host. In the case of sending Data, the program goes first to the Collision routine.

Sending Data to the Host

Data is sent only in response to a Talk Command. For every data bit cell, the line is tested to go high at the proper time. If the line is still low, a collision has occurred. When a collision is detected, a collision flag is set, and the program aborts to look for a Command signal sequence.

FIGURE 4: TYPICAL TRANSACTION WITH SERVICE REQUEST



© 1997 Microchip Technology Inc.

IMPLEMENTATION

Hardware

The hardware of this circuit is fairly simple. The circuit is powered via the +5V and GND wires of the ADB cable. The ADB I/O wire is connected to pin RA0 with a pull-up resistor to 5V. The T0CKI pin is tied to GND. The Master Clear ($\overline{MCLR}$) pin is tied to 5V.

This circuit uses a 4 MHz crystal as a timing reference, but higher values may be substituted. The software is designed to accommodate higher frequencies.

A pushbutton switch is used as the single key of the "keyboard." One side is connected to port RB1 with a pull-up resistor to 5V, and the other side to GND. An LED is used to indicate that the 'key' has been pressed, with the positive side connected to pin RB0 and the negative side to GND.

Software

The program designated as "Application Tasks," has two sections, one is setup to switch between a protocol support task for the ADB signal decode and processing, and the other section is the Application Task, in this case a single key "keyboard" routine. The ADB protocol task has priority. The first section of the code is the ADB protocol task, the second section is the Application Task, "Keyboard." The two tasks communicate through flags which indicate that data needs to be sent, or that data has been received.

The Keyboard Task is run at two times; 1) during the Attention Signal, 2) between the end of the Data Stop Bit and the beginning of the Attention Signal. The Keyboard Tasks is given up to 500 µs during the Attention Signal, and 900 µs during the time between the end of the Data Stop Bit and the beginning of the Attention Signal. It is important to note here that the other tasks MUST NOT AFFECT TMR0 or the ADB time variable that the Attention Signal is using to keep track of the RTCC.

Timing

Timing is accomplished by first loading a constant into a time variable. This constant represents the maximum limit for the current routine, which may not necessarily be the maximum timing range for the current Signal. The TMR0 value is loaded into the working register, and subtracted from the time variable. The Carry bit of the STATUS register is tested to see if it is set or clear. If the bit is clear, the current timing limit has been exceeded. Further action is taken based on this status. It is important to keep the constant away from 255, or rollover may occur, giving inaccurate results. The prescaler is applied to the TMR0 as necessary.

The following are the timing ranges used by this program for ADB signals:

Reset	> 824 µs
Attention	776-824 µs
Sync	72 µs
Bit Cell	Up to 104 µs
'1' Bit low time	< 50 µs
'0' Bit low time	> 50 < 72 µs
Stop bit	0 Bit
Stop to Start (Tlt)	140-260 µs
Service Request (Srq)	300 µs

Note: The range of values given for 0 Bit, 1 Bit and Tlt timing are slightly wider than those given in the ADB specification.

How Address Conflicts are Resolved

During the start-up process the host sends a "Talk Register 3" command to each device address, and waits for a response. When a device recognizes that the Host issued a "Talk Register 3" command, it responds by sending a random address. During the transfer of each Bit Cell of the random address the signal line is monitored for the expected signal level. If the signal is not what is expected there is an address conflict. If the address is sent successfully, the host will respond with a Listen Command to that device. The command will have a new Device Address to which that device will move. The device then only responds to commands at the new address.

If there is a conflict, where two devices have the same default address, and respond at the same time, the device that finds the line low when it expects it to be high, immediately stops transmitting because it has determined that a collision has occurred. The device which detected the collision marks its address as unmovable and therefore ignores the address move Command, a Listen Register 3 Command. The device maintains the unmovable address condition until it has executed a successful response to the Talk Register 3 Command.

The host continues sending a Talk Register 3 Command at the same address until there is a time-out and no device responds. This is how conflicts are resolved when more than one device has the same address; for example, if two keyboards are connected.

© 1997 Microchip Technology Inc.

Program Sequence:

Words in parenthesis, ( ), accompanying the TITLES are Labels of procedures in the corresponding code.

Start-up / IDLE (`Start`)

Start by configuring the ADB pin on PORTA and the Switch Pin on PORTB as inputs, and tri-stating the rest of PORTA and PORTB as outputs.

INITIALIZE DEFAULT VALUES WHEN THE LINE IS HIGH (`Reset`)

Look for the line to be high, and when it is, clear or initialize registers to default values.

LOOK FOR ATTENTION OR RESET (`AttnSig`)

Look for the line to go low, when it does, clear TMR0 and time how long it is low. An Attention Signal has occurred if the line went high between 776 and 824 µs. If the low time is measured to less than 776 µs, another signal has occurred and the program aborts, looking for the Attention Signal again. When the low time is measured to greater than 824 µs, the program interprets this timing as a Reset Signal. The program starts over again, waiting for the line to be high, and when it is, performs a Reset initialization.

Note: The keyboard task is performed during the Attention Signal (Task_2).

LOOK FOR SYNC SIGNAL (`SyncSig`; calls `Srq`)

The Sync Signal is the high time between the rising edge of the Attention Signal and the falling edge of the first bit of the Command.

GET THE COMMAND (Command; calls `Get_Bit`)

Look for the Command; a combination of eight '0' and '1' bits. The MSb is sent first. This is achieved by calling the `Get_Bit` routine, which checks whether the maximum Bit Cell time is exceeded, if not, it looks for the rising edge at the end of the bit. When the bit is received, it is rotated into a variable, and the end of the bit cell is expected. When the falling edge of the next bit is detected, the routine clears TMR0 and returns to Command, which calls `Get_Bit` again until all 8-bits of the Command have been received.

ISSUE A SERVICE REQUEST IF NECESSARY (`Srq`)

If data needs to be sent to the Host, a Service Request (`Srq`) is issued by holding the line low while the Stop Bit is being received during the Stop-to-Start time (`Tlt`) which is between the end of the Command Stop Bit and the beginning of the Data Start Bit.

LOOK FOR STOP BIT (`CmdStop`)

Look for the Stop Bit (a '0' bit of 65 µs) that comes after the last Command Byte.

INTERPRET THE COMMAND (`AddrChk`)

After the command has been received, determine if the address belongs to this device. If the address is not for this device, determine if the command is global for all devices and if so, do that command. If this is not a Global/Reserved Command, call the Service Request (Srq) Routine to see if an Srq should to be issued to the Host, and do so if necessary, then return to get the Attn Signal. If the Address is for this device determine whether it is a Talk, Listen, or Flush Command, and go to the specified Command routine.

SENDING DATA (Talk; calls `Tlt`)

If the command was interpreted to be a Talk Command addressed to this device, call the Stop-to-Start Time (Tlt) routine. When the `Tlt` routine has completed, determine if this is a Talk Register 3 Command. If so, return a Random Address as part of the two bytes sent to the Host. If this is not a Talk Register 3 Command, determine if data needs to be sent. If so, send the Data Start Bit (a '1'), two bytes of data from the indicated register, and a Stop Bit (a '0'). If not, abort to the Attention Signal. If at any time the transmission of Data is interrupted, abort to the Collision routine. Only after a complete transmission should the flags be cleared indicating a successful transmission.

Note: The ADB Specification indicates data may be between two and eight bytes long. The limitations of the PIC16C54/55/56 parts allow only two bytes of data to be sent by this program due to limited register space. If more than two bytes of data must be sent, use the PIC16C57.

RECEIVING DATA (Listen; calls `Tlt`)

If the command was interpreted to be a Listen Command addressed to this device, call the Stop-to-Start Time (`Tlt`) routine. When the `Tlt` routine has completed, receive the rest of the Data Start Bit, 2 Data Bytes, and Data Stop Bit. When the data has been received, determine whether this is a Listen Register 3 Command. If this is a Listen Register 3 Command, interpret what the command is. If this is a conditional Address Change Command, determine if this Device's Address is moveable at this time. If not, abort to the Attention Signal. If so, change the device to the new address and go run the Second Application Task. If this is not a Listen Register 3 Command, move the data into the specified register and go run the Second Application Task.

© 1997 Microchip Technology Inc.

LOOK FOR THE STOP TO START TIME (Tlt)

After the Command and Stop Bit, the Talk or Listen routines call the Tlt routine. Tlt looks for the line to go low. If the line went low before the minimum Tlt Time, see if this is a Talk Command. If this is a Talk Command, abort to the Collision routine. If this is a Listen Command, abort to the Attention Signal.

If the minimum Tlt time passes and the line is high, see if the Talk routine called the Tlt, if so, go wait for until the middle of the Tlt, then return to the Talk routine to send the Data Start Bit, Data Bytes, and Stop Bit. If at any time the line goes low during the Tlt and the Talk routine called it, abort to the Collision routine.

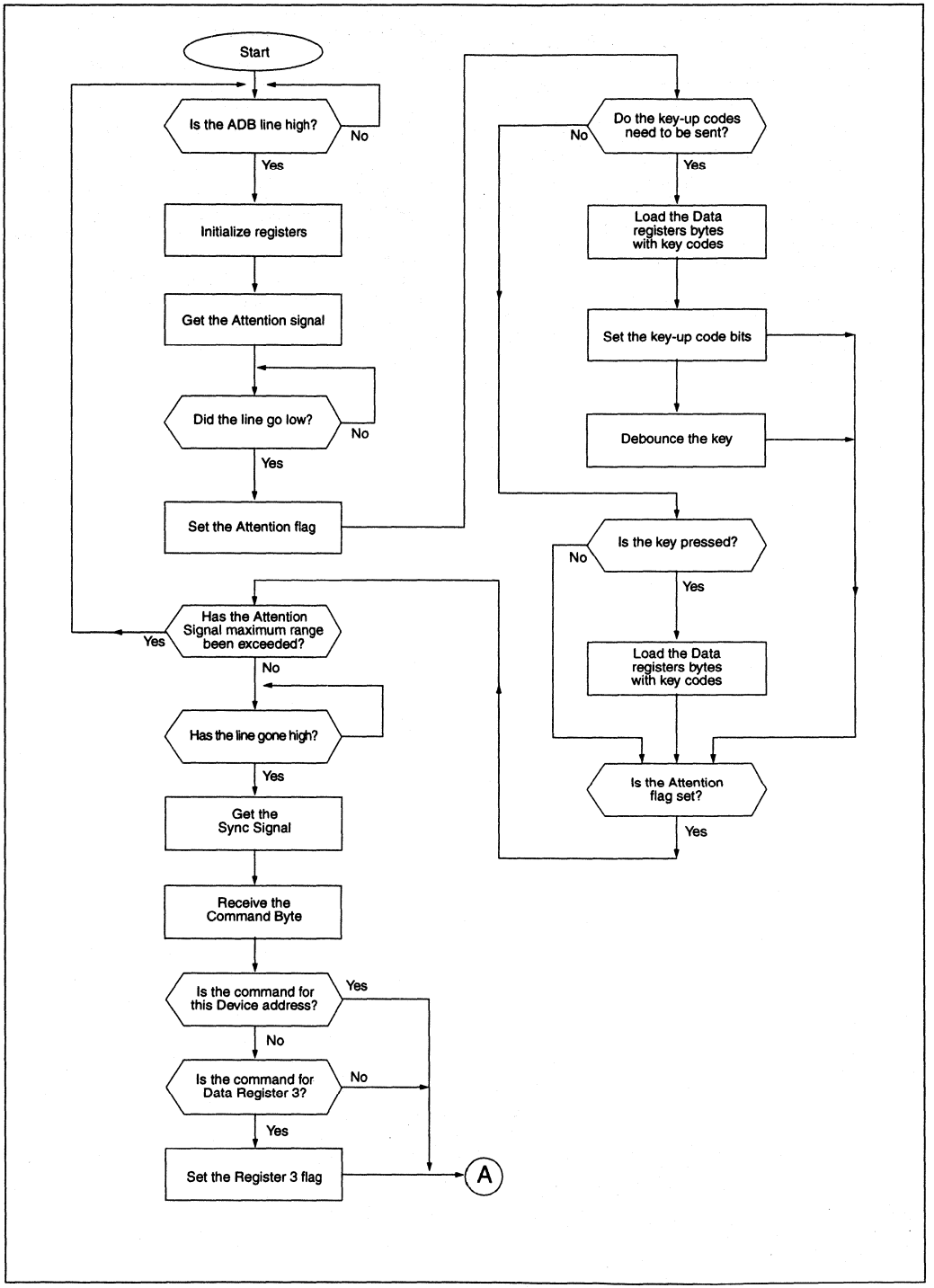
If the Listen routine did call Tlt, look for the line to go low at the beginning of the Data Start Bit. When the line goes low, return for the rest of the Start Bit. If the line doesn't go low before the maximum Tlt time is up, abort to the Attention Signal.

THE KEYBOARD TASK IS PERFORMED BETWEEN THE END OF THE DATA STOP BIT AND THE ATTENTION SIGNAL (Task_2)

The Keyboard Task checks to see if the key has been pressed. When the key is pressed, indication flags are set and an LED is turned on until the key has been debounced. The flags allow the key to be debounced, Srq(s) to be sent to the Host, and indicate to the Talk routine that Data needs to be sent. Two bytes of data are loaded into Register 0 representing a key-down code and a flag is set indicating to the ADB task that data needs be sent to the host. When the key-down codes have been sent, the key-up codes are loaded into Register 0. When the key-up codes have been sent and the key has been debounced, the flags are cleared. The final routine of Task_2 decides whether to return to the beginning or middle of the Attention Signal.

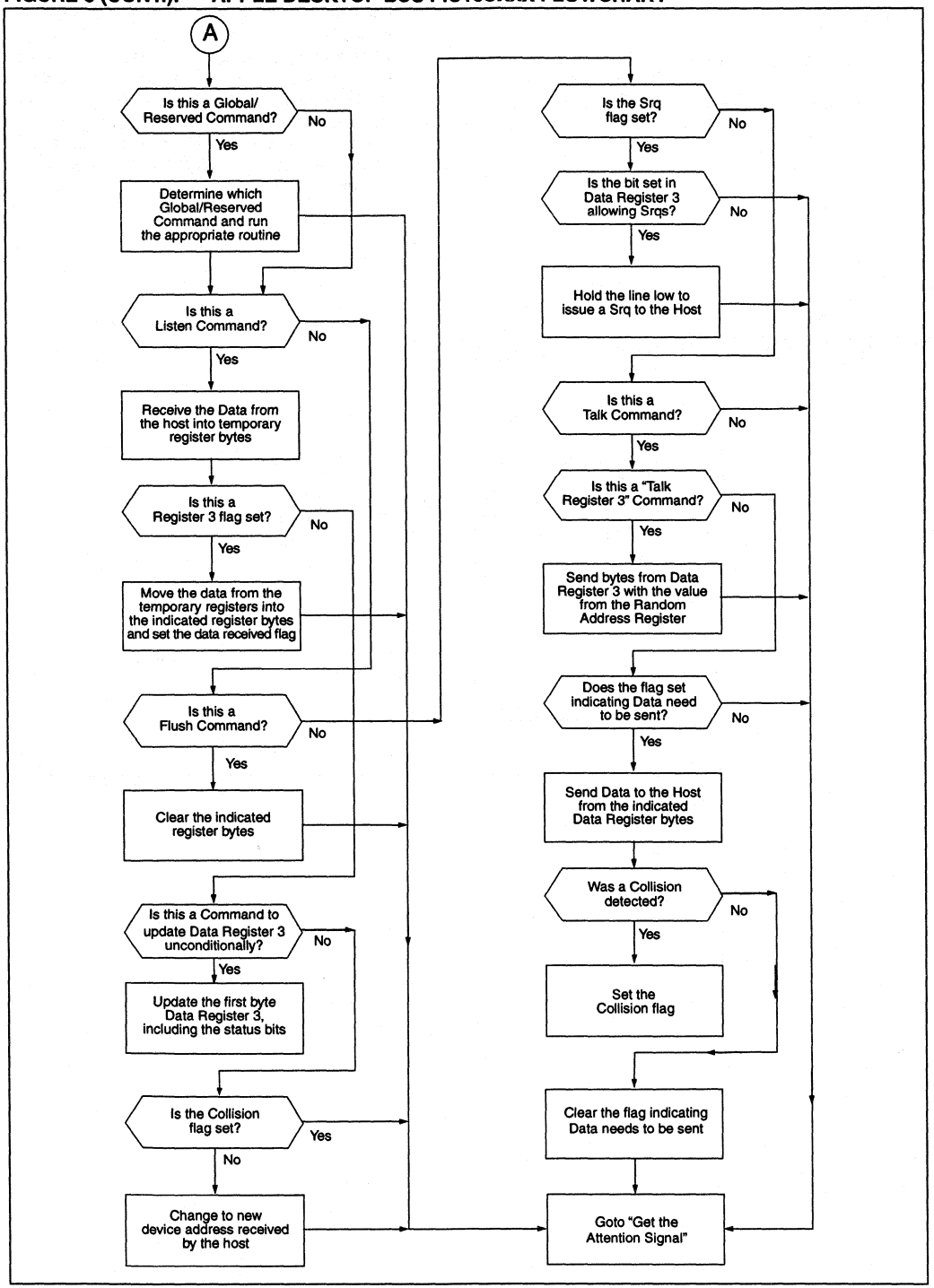
© 1997 Microchip Technology Inc.

FIGURE 5: APPLE DESKTOP BUS PIC16CXXX FLOWCHART



© 1997 Microchip Technology Inc.

FIGURE 5 (CONT.): APPLE DESKTOP BUS PIC16CXXX FLOWCHART

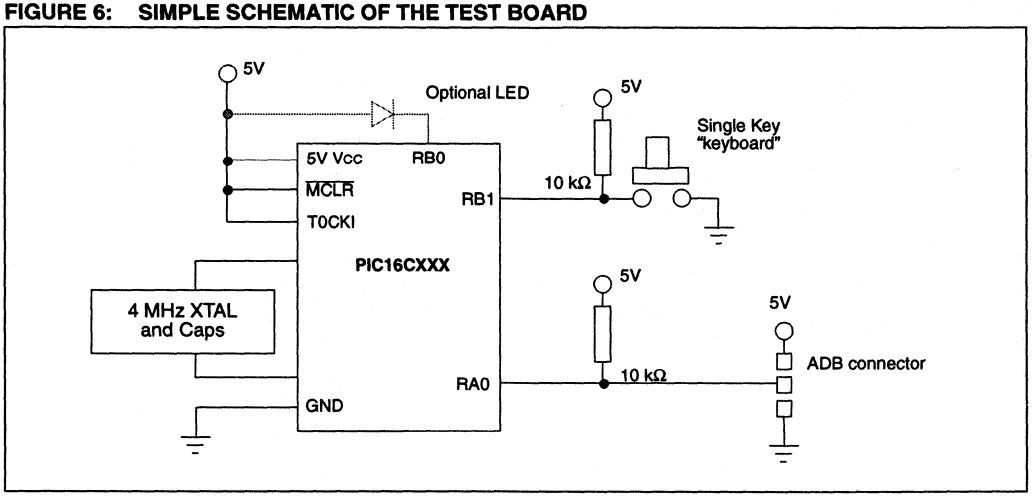


© 1997 Microchip Technology Inc.

SUGGESTIONS ABOUT MODIFYING THE CODE

1. If high crystal frequencies are used, a divider equate (equ) at the beginning of the timing section of the equates allows an easy adaptation for all established timing definitions.
2. The second application task may occur as a communication task with another PIC16CXXX device by using the three other I/O lines on PORTA, although test code for this has not yet been written. Two of the lines would be used as ready-to-send (one for each PIC16CXXX). The third would be used as a data line, using low signals as '0' bits, and high signals as '1' bits. Additionally, all eight lines on PORTB may be used as well.

FIGURE 6: SIMPLE SCHEMATIC OF THE TEST BOARD



© 1997 Microchip Technology Inc.

RESOURCES

Apple Publications and Support Software

MacTech Magazine (formerly MacTutor) is a publication dedicated to supporting the Macintosh. They have had several articles regarding the Apple Desktop Bus. They publish a CD-ROM that contains all of their articles from 1984 to 1992. Also, single disks are available (ask for #42).

MacTech Magazine can be contacted at:

P.O. Box 250055
Los Angeles, CA 90025-9555
310 575-4343 FAX 310 575-0925
Applelink: MACTECHMAG
Internet: info@xplain.com

Apple licenses the ADB technology. They can be contacted at:

20525 Mariani Ave.
Cupertino, CA 95014
Attn: Software Licensing

- Apple Keyboard, extended, specification drawing #062-0168-A.
- Apple Desktop specification drawing # 062-0267-E.
- Apple Desktop connector, plug, Mini DIN drawing #519-032X-A.
- Engineering Specification, Macintosh transceiver interface, ADB drawing #062-2012-A.
- Apple keyboard, specification drawing #062-0169-A.
- Developer CD series, Tool Chest Edition, August 1993 contains:
  - Folder = Tool Chest: Devices and Hardware: Apple Desktop Bus
  - ADB Analyzer
  - ADB Parser (most complete environment)
  - ADB Lister
  - ADB ReInit
  - ADB Tablet code samples

WFT Electronics offers free assistance in procuring necessary ADB info. Contact Gus Calabrese, Rob McCall, Dave Evink at:

4555 E. 16th Ave.
Denver, CO 80220
303 321-1119 FAX 303-321-1119 Applelink: WFT
Internet: Gus_Calabrese@onenet-bbs.orgA

AUTHOR / CREDITS

Rob McCall developed the majority of the PIC16CXX ADB code. He also wrote most of the application note. Gus Calabrese, Dave Evink, and Curt Apperson supported this effort. Dave works with Gus, Rob, and Curt in developing a variety of embedded processor products.

Contact Gus Calabrese, Rob McCall, Dave Evink, Curt Apperson at:

WFT Electronics
4555 E. 16th Ave.
Denver, CO 80220
303 321-1119 FAX 303-321-1119 Applelink: WFT
Internet: Gus_Calabrese@onenet-bbs.org

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: ADB.ASM

```
MPASM 01.40 Released              ADB.ASM   1-16-1997  17:26:35         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C56, n = 66,   c=132,  E=0,  N=60
                     00002 ;
                     00003 ;*****************************************************************A113-0004
                     00004 ;
                     00005 ; ADB.ASM *** This program is for PIC16C5x microcontrollers:
                     00006 ;
                     00007 ;       Program:          ADB.ASM
                     00008 ;       Revision Date:
                     00009 ;                         1-16-97      Compatibility with MPASMWIN 1.40
                     00010 ;
                     00011 ;*******************************************************************************
                     00012 ;
                     00013 ;**TESTING - The purpose of this program is to emulate a keyboard that
                     00014 ;  is Apple Desktop Bus (ADB) based.  The program allows the PIC to
                     00015 ;  appear to the Macintosh computer as a keyboard with a single key.
                     00016 ;  The code isdesigned to easily modify the device address to make the
                     00017 ;  PIC appear as another ADB device,which has its own proprietary
                     00018 ;  functions.
                     00019 ;*******************************************************************************
                     00020 ;
                     00021 ; OVERVIEW OF ENTIRE PROGRAM:
                     00022 ; This program is setup to switch between a communication task with the
                     00023 ; the Apple Desktop Bus (ADB), and another application task.
                     00024 ; The ADB communication task has priority.
                     00025 ; All communication with the ADB is done using a single i/o line to
                     00026 ; the PIC, line RA0 on Port A.
                     00027 ; The second application may occur as a communication task with
                     00028 ; another PIC  chip as follows:
                     00029 ; Communication with the second PIC may be achieved by using the three
                     00030 ; other i/o lines on Port A.  Two of the lines would be used as
                     00031 ; ready-to-send (one for each PIC).  The third would be used as a data
                     00032 ; line, using low signals as 0 bits, and high signals as 1 bits.
                     00033 ; Additionally, all eight lines on PORTB may be used as well.
                     00034 ;
                     00035 ;*********************  ADB COMMUNICATION TASK  ******************
                     00036 ;
                     00037 ;**** A BRIEF DESCRIPTION OF THE ADB COMMUNICATION SEQUENCE:
                     00038 ;
                     00039 ;  STARTUP    ------- initialize the TMR0 prescaler & Tri-States PORTA
                     00040 ;
                     00041 ;  Look for the following signals and/or take appropriate actions:
                     00042 ;  RESET ----------- a high line, then initialize default register values
                     00043 ;  ATTENTION ------- Attention signal, (there is enough time during this
                     00044 ;                     signal to allow other tasks to be performed)
                     00045 ;  COMMAND --------- 8 Command bits followed by a Stop Bit
                     00046 ;  INTERPRET ------- Decide whether the Host is addressing this Device,
                     00047 ;                     if so, decide what Command the Host issued
                     00048 ;                     if not, see if the Command is global to all Devices,
                     00049 ;                     also determine if the other Application needs to
                     00050 ;                     issue a Service to the Host.
                     00051 ;  Tlt ------------ The time between the Stop bit of the Command byte and
                     00052 ;                     the Start Time of the data being received/sent. Also
                     00053 ;                     referred to as Stop to Start Time.
                     00054 ;  SERVICE REQUEST - in order for a Device to alert the Host that it has
```


© 1997 Microchip Technology Inc.

```
00055 ;           data to send, the line is held down after the Command Stop
00056 ;           Bit (continuing on from the Tlt).
00057 ;   DATA -- a Data Start Bit, followed by 2 Data Bytes (up to as
00058 ;           many as 8 Bytes), and a final Stop Bit
00059 ;
00060*****************************************************************************
00061 ;
00062 ; THE FOLLOWING IS A MORE DETAILED DESCRIPTION OF THE PROGRAM SEQUENCE:
00063 ;
00064 ; NOTE: words in parenthesis accompanying the TITLES are Labels of
00065 ; procedures corresponding in the code below.
00066 ;
00067 ;*** STARTUP / IDLE ***  (Start)  ***
00068 ; Startup by setting the ADB pin on PORTA as an input and tri-stating the
00069 ; rest as outputs. The routine then goes to the Reset routine.
00070 ; NOTE: For testing, pin RB1 is is set as an input, and the rest of PORTB
00071 ; is tri-stated as an output.
00072 ;
00073 ;*** INITIALIZE DEFAULT VALUES WHEN THE LINE IS HIGH ***  (Reset)  ***
00074 ; Look for the line to be high, and when it is, initializes the
00075 ; registers to default values.
00076 ;
00077 ;*** LOOK FOR ATTENTION OR RESET ***  (AttnSig)  ***
00078 ; Look for the line to go low, when it does, clear the TMR0 and time how
00079 ; long it's low.
00080 ; An Attention Signal has occurred when the line goes high between 776 and
00081 ; 824 usecs.
00082 ; If the low time is measured less than 776 usecs, another signal has
00083 ; occurred and the program aborts, looking for the Attention Signal
00084 ; again. When the low time is measured greater than 824 usecs, the program
00085 ; interprets this timing as a Reset Signal.  The program starts over
00086 ; again, waiting for the line to be high, and when it is, performs a
00087 ; Reset initialization.
00088 ;*** OTHER APPLICATION TASKS MAY BE PERFORMED DURING
00089 ;                              THE ATTENTION SIGNAL *** (Task_2) ***
00090 ; The time during which the Attention signal takes place allows a second
00091 ; state to occur. The other task(s) is/are given up to 500 usecs during
00092 ; the  Attention Signal (900 usecs are given to the 2nd Task during the
00093 ; time between the end of the Data Stop Bit and the beginning of
00094 ; the Attention Signal.
00095 ; It is important to note here that the other task(s) MUST NOT AFFECT
00096 ; THE Timer0 or the time variable (TimeVar) that the Attention Signal is
00097 ; using to keep track of the TMR0.
00098 ;
00099 ;********** NOTE:
00100 ; If at any time during the detection of the Signals below, the line is
00101 ; low or high outside of timing ranges, the routine aborts to see if an
00102 ; Attention  or Reset signal has been issued by the Host, or, in the
00103 ; case of sending  Data, to the Collision routine.
00104 ;
00105 ;*** LOOK FOR SYNC SIGNAL ***  (SyncSig)  ***
00106 ; The Sync Signal is the high time between the rising edge of the
00107 ; Attention Signal and the falling edge of the first bit of the Command.
00108 ;
00109 ;*** GET THE COMMAND ***  (Command; calls GetBit)  ***
00110 ; Look for the Command, a combination of eight 0 and 1 bits, MSB sent
00111 ; first. This is achieved by calling a the GetBit routine which checks
00112 ; whether the maximum time is exceeded, if not, looks for the rising edge
00113 ; at the end of the bit.  When the bit is received, it is rotated into a
00114 ; variable, and the end of the bit cell is expected.  When the falling
00115 ; edge of the next bit is detected, the routine clears TMR0 and
00116 ; returns to Command, which calls GetBit again until all 8 bits of the
00117 ; Command have been received.
00118 ;*** ISSUE A SERVICE REQUEST IF NECESSARY ***  (Srq)  ***
00119 ; If data needs to be sent to the Host, issue a Service Request (Srq) by
00120 ; holding the line low while the Stop Bit is being recieved, during the
```


© 1997 Microchip Technology Inc.

```
00121 ;Stop-to-Start time (Tlt) between the end of the Command Stop bit and
00122 ;the beginning of the Data Start Bit.
00123 ;
00124 ;*** LOOK FOR STOP BIT ***  (CmdStop)
00125 ;Look for the Stop Bit (a 0 bit of 65 usecs) that comes after the last
00126 ;Command Byte.
00127 ;
00128 ;*** INTERPRET THE COMMAND ***  (AddrChk)  ***
00129 ;After the Command has been received, determine if the Address belongs to
00130 ;this Device.
00131 ;If the Address is not for this Device determine if the command is
00132 ;global for all Devices and if so, do that command.
00133 ;If this is not a Global/Reserved command, call the Service Request (Srq)
00134 ;routine to see if an Srq should to be issued to the Host, and do so if
00135 ;necessary, then return to get the Attn Signal.
00136 ;If the Address is for this Device determine whether it is a Talk,
00137 ;Listen, or Flush Command, and go to the specified command routine.
00138 ;
00139 ;**IF COMMAND IS TALK OR LISTEN, LOOK FOR STOP TO START TIME ** (Tlt) **
00140 ;After the Command and Stop Bit (a 0 bit) the Talk or Listen routine
00141 ;calls the Tlt routine:
00142 ;look for the line to go low,
00143 ;if the line went low before the Min. Tlt Time, see if this is a Talk
00144 ;Command if this is a Talk Command, abort to the Collision routine
00145 ;if this is a Listen Command, abort to the Attention Signal
00146 ;if the Min. Tlt time passes & the line is high,
00147 ;see the Talk routine called the Tlt,
00148 ;if so, go wait for until the middle of the Tlt, then return to
00149 ;Talk to send the Data Start Bit, Data Bytes, and Stop Bit.
00150 ;if at any time the line goes low during the Tlt, abort to the
00151 ;Collision routine
00152 ;if Listen called the Tlt,
00153 ;look for the line to go low as the beginning of the Data Start Bit
00154 ;if the line goes low, return for the rest of the Start Bit
00155 ;if the line doesn't go low before the Max. Tlt time,
00156 ;abort to the Attention Signal
00157 ;
00158 ;*** SENDING DATA ***  (Talk)  ***
00159 ;If the Command was interpreted to be a Talk Command addressed to this
00160 ;Device, call the Stop-to-Start Time (Tlt) routine.
00161 ;When the Tlt routine has completed, determine if this is a Talk Register
00162 ;3 Command.  If so, and if so, return a Random Address as part of the
00163 ;two bytes sent to the Host.
00164 ;if this is not a Talk Register 3 Command, determine if Data needs to be
00165 ;sent.  If so, send the Data Start Bit (a '1'), two bytes of Data,
00166 ;and a Stop Bit (a '0').  If not, abort to the Attention Signal
00167 ;If at any time the transmission of Data is interrupted, abort to the
00168 ;Collision routine.  Only after a complete transmission should the
00169 ;flags be cleared indicating a successful transmission.
00170 ;NOTE:  The ADB Spec. indicates data may be between 2 and 8 bytes long.
00171 ;The limitations of the PIC 16C54/55/56 parts allow only 2 bytes of data
00172 ;to be sent by this program due to limited register space. If more than
00173 ;2 bytes of data must be sent, use the PIC16C57.
00174 ;
00175 ;*** RECEIVING DATA ***  (Listen)  ***
00176 ;If the Command was interpreted to be a Listen Command addressed to this
00177 ;Device, call the Stop-to-Start Time (Tlt) routine.
00178 ;When the Tlt routine has completed, receive the rest of the Data
00179 ;Start Bit, 2 Data Bytes, and Data Stop Bit.
00180 ;When the Data has been received, determine whether this is a Listen
00181 ;Register 3 Command.
00182 ;if this is a Listen Register 3 Command, interpret what the Command
00183 ;is.  If this is a conditional Address change command, determine if
00184 ;this Device's Address is moveable at this time.  If not, abort to the
00185 ;Attention Signal.  If so, change the Device to the new Address and
00186 ;go run the Second Application Task.
```


© 1997 Microchip Technology Inc.

```
                00187 ; if this is not a Listen Register 3 Command, move the Data into the
                00188 ; specified register and go run the Second Application Task.
                00189 ;
                00190 ;*************************************************************************
                00191 ;
                00192 ;*** TIMING ALGORITHM ***
                00193 ; Timing for ADB signals is done by clearing the TMR0, loading a constant
                00194 ; into  a time variable, subtracting the TMR0 from the variable,
                00195 ; This process is looped until the either the Carry Bit in the Status
                00196 ; Register is clear, indicating the amount of time in the time variable
                00197 ; has elapsed, or the condition of the data line has been met.
                00198 ; If the line goes high or low at an inappropriate time, an error has
                00199 ; occurred, and the current operation should be aborted.
                00200 ;
                00201 ; NOTE: The minimum and maximum values given below for 0 bit, 1 bit, and
                00202 ;       Tlt timing are slightly shorter and longer than those given in
                00203 ;       the ADB specification.
                00204 ;  The following are the timing ranges used
                00205 ;  by this program for ADB signals:
                00206 ;  Reset ........Greater Than 824  usecs
                00207 ;  Attention..............776-824  usecs
                00208 ;  Sync.....................72   usecs
                00209 ;  Bit Cell.............Up to 104  usecs
                00210 ;      1 Bit...............32-40   usecs
                00211 ;      0 Bit...............60-72   usecs
                00212 ;  Stop bit................60-72   usecs
                00213 ;  Stop to Start (Tlt)....140-260  usecs
                00214 ;  Service Request (Srq).....300  usecs
                00215 ;
                00216 ;
                00217 ; A SOMEWHAT GRAPHICAL REPRESENTATION OF THE TIMING SIGNAL RANGES (in
                00218 ; usecs):
                00219 ;  |-------|--------|--------|
                00220 ;         30-40    60-70     100
                00221 ;         1 Bit    0 Bit    End of Bit Cell
                00222 ;
                00223 ;  |--------------|----------|
                00224 ;                140-260       300
                00225 ;                  Tlt         Srq1
                00226 ;
                00227 ;  |----------------------------------|--------|---|--------------->
                00228 ;  0                                  776      824  Greater than 824....
                00229 ;  Signal invalid in this area--------|  AttnSig    Reset
                00230 ;
                00231 ;*************************************************************************
                00232 ;
                00233 ;********************   THE PROGRAM BEGINS HERE   **********************
                00234
                00235     include "p16c5X.inc"   ; default EQUates for the PIC registers
                00001         LIST
                00002 ; P16C5X.INC Standard Header File, Version 3.30 Microchip Technology
                00224         LIST
                00236
000001FF        00237 PIC54   equ     1FFh    ; Define the Reset Vector for 16c54.
                00238
00000000        00239 NULL    equ     00h     ; used for returning nothing from a called routine
                00240
00000000        00241 LSB     equ     00h     ; Least Significant Bit
00000007        00242 MSB     equ     07h     ; Most Significant Bit
                00243
00000000        00244 FALSE   equ     00h     ; For Boolean tests
00000001        00245 TRUE    equ     01h
                00246
                00247     include "adb.equ"  ; ADB EQUates
                00001 ;************************* ADB.EQU Header-sets up EQUates *************
                00002
```

```
                    00003 ;*** TESTING *** BITS USED IN TESTING FOR I/O
                    00004
                    00005 ; *** BOOLEANS USED TO SELECT PART BEING USED
                    00006                          ; Only One Part May Be selected at a time
00000000            00007 C54      equ     FALSE   ;TRUE
00000000            00008 C55      equ     FALSE
00000001            00009 C56      equ     TRUE    ;FALSE
00000000            00010 C57      equ     FALSE
                    00011
00000000            00012 LED      equ     00h     ; ***AN LED ON LINE RB0 INDICATES SWITCH PRESSED
00000001            00013 Switch   equ     01h     ; ***'Switch' USED FOR A SWITCH ON LINE RB1 AND
                    00014                          ; *** AS A FLAG IN FLAGS2 FOR DEBOUNCING
                    00015
00000038            00016 SHIFT    equ     38h
00000012            00017 BANG     equ     12h
                    00018
00000008            00019 DEBOUNC equ      08h  ; *** #OF TIMES TO LOOP TO ALLOW DEBOUNCE OF SWITCH
                    00020
                    00021
                    00022
                    00023
                    00024 ; *** BIT ASSIGNMENTS FOR I/O LINES & TRI-STATING
                    00025
00000000            00026 ADB      equ     00h     ; Line used for ADB - pin XX (16C54)
00000001            00027 RA1      equ     01h     ; May be used as a Clock line TO another PIC
00000002            00028 RA2      equ     02h     ; May be used as a Clock line FROM another PIC
00000003            00029 RA3      equ     03h     ; May be used as a Data line between two PICs
                    00030
00000001            00031 TRI_IN   equ     01h     ; tri-state for ADB pin as input
00000000            00032 TRI_OUT equ      00h     ; tri-state for ADB pin as output
                    00033
                    00034
                    00035 ;*** MISC. CONSTANTS
                    00036
00000008            00037 BYTE     equ     08h     ; Receive 8 bits in Command; count from 8 to 0
00000002            00038 DEF_ADD equ      02h     ; default device address to start with (kybd)
00000003            00039 DEF_HND equ      03h     ; default Handler Id. to start with (std. kybd)
00000008            00040 OFFSET   equ     08h     ; offset to RAM address of the array of ADB
                    00041                          ; Data storage registers
                    00042
                    00043
                    00044 ;*** COMMAND MASKS:          MASK BITS FROM COMMAND REGISTER FOR:
                    00045
0000000F            00046 DEVMASK equ   0Fh     ;lower nibble holds Command (Talk, etc.) & Reg. #
000000F0            00047 ADDRMSK equ   0F0h    ;upper nibble holds the Device Address Number
0000000F            00048 CMDNIBL equ   0Fh     ;Command nibble from the address
0000000C            00049 CMDTYPE equ   0Ch     ;Upper 2 Command bits indicate Talk, Listen, etc.
00000003            00050 REGMASK equ   03h     ;Data Register Number bits from Command Nibble
0000001F            00051 FSRMASK equ   1Fh     ;FSR bits from the Command Nibble for RAM Address
                    00052
                    00053
                    00054 ;*** DATA COMMAND MASKS:    MASK DATA REGISTER 3a FOR:
                    00055
0000000F            00056 LOW_NBL equ     0Fh   ; Lower nibble from the 1st Data byte
000000F0            00057 HI_NIBL equ    0F0h   ; Upper nibble from the 1st Data byte
                    00058
                    00059
                    00060 ;*** CONSTANTS FOR MASKING OUT COMMAND NIBBLES (C_ indicates Command)
                    00061
                    00062                          ;   used to XOR if this is a:
0000000C            00063 C_TALK   equ     0Ch     ;   Talk Command
00000008            00064 C_LISTN equ      08h     ;   Listen Command
00000000            00065 C_RESET equ      00h     ;   Reset Command
00000001            00066 C_FLUSH equ      01h     ;   Flush Command
00000004            00067 C_RES_1 equ      04h     ;   Reserved Command 1
00000002            00068 C_RES_2 equ      02h     ;   Reserved Command 2
```


© 1997 Microchip Technology Inc.

```
00000003          00069 C_RES_3 equ     03h   ;   Reserved Command 3
                  00070
                  00071
                  00072 ;*** DATA HANDLER ID MASKS: MASK DATA REGISTER 3b FOR:
                  00073
000000FF          00074 SELFTST equ    0FFh   ; Self-Test mode
00000000          00075 LISTEN1 equ      0h   ; unconditional address change
000000FE          00076 LISTEN2 equ    0FEh   ; address change if no collision detected
000000FD          00077 DEV_ACT equ    0FDh   ; address change if device activator is depressed
                  00078
                  00079
                  00080 ;BITS USED IN THE UPPER NIBBLE OF REGISTER 3a FOR SPECIAL ADB STATUS BITS
                  00081
00000004          00082 Resrvd3 equ     04h   ; reserved (Always 0)
00000005          00083 Srq_Bit equ     05h   ; determines if Host will accept Service Requests
00000006          00084 ExpEvnt equ     06h   ; indicates an Exceptional Event should take place
00000007          00085 Always0 equ     07h   ; always set to 0
                  00086
                  00087
                  00088 ;ADB FLAG BITS IN THE "FLAGS1" REGISTER (F1 indicates 1st Flags register)
                  00089
00000000          00090 F1Attn  equ     00h   ; set to know if 2nd Task taking place during Attn
00000001          00091 F1Reg3  equ     01h   ; Register 3 is being addressed
00000002          00092 F1Talk  equ     02h   ; indicates to Tlt routine this is a Talk Command
00000003          00093 F1Stop  equ     03h   ; set to indicate the Data Stop Bit is being sent
00000004          00094 F1Lstn  equ     04h   ; indicates to Tlt routine this is a Listen Command
00000005          00095 F1Sent1 equ     05h   ; 1st byte of Data Register has been sent
00000006          00096 F1Rcvd1 equ     06h   ; 1st byte of Data Register has been received
00000007          00097 F1Cllsn equ     07h   ; set to indicate that a collision occurred
                  00098
                  00099
                  00100 ;*** FLAG BITS IN THE "FLAGS2" REGISTER (F2 indicates 2nd Flags register)
                  00101
00000000          00102 F2Srq   equ     00h  ; indicate that Srq should be issued
                  00103 ;               01h Switch, defined above for PORT_B, also used as a Flag
00000002          00104 F2DActv equ     02h  ; change address if Device Activator is Depressed
00000003          00105 F2STest equ     03h  ; set to indicate a device Self Test to be performed
00000004          00106 F2SFail equ     04h  ; set to indicate that the Device Self-Test Failed
00000005          00107 F2DRcvd equ     05h  ; set when data is received for 2nd Application Task
00000006          00108 F2DSend equ     06h  ; set to indicate to Talk that Data needs to be sent
00000007          00109 F2DMore equ     07h  ;set in 2nd Task to indicate Data remains to be sent
                  00110
                  00111
                  00112 ;*** TIMING DEFINITIONS
                  00113                      ; These values currently used for clock at 4Mhz:
00000004          00114 PrSclr1 equ     .4   ;   this is used when TMR0 is being prescaled
00000001          00115 PrSclr2 equ     .1   ;   this is used when TMR0 is not prescaled
                  00116
000000C2          00117 ATT_MIN equ .776/PrSclr1 ; Attn lower limit:800 - 3% tolerance=776 usecs
000000CE          00118 ATT_MAX equ .824/PrSclr1 ; Attn upper limit:800 + 3% tolerance=824 usecs
0000007D          00119 TSK2MIN equ .500/PrSclr1 ; time given to 2nd Task during Attn Signal
000000E1          00120 TSK2MAX equ .900/PrSclr1 ;time given to 2nd Task after Data Sent/Received
00000048          00121 SYNC    equ .72/PrSclr2 ;Sync with extra tolerance after Attn detect
00000032          00122 BIT_TST equ .50/PrSclr2  ; if time is < 50 = 1 bit, & > 50 = 0 bit
00000048          00123 MAX_BIT equ .72/PrSclr2  ; Maximum time line can be low for a bit
00000068          00124 BITCELL equ .104/PrSclr2 ; Maximum time for a bit cell = 104 usecs
0000008C          00125 TLT_MIN equ .140/PrSclr2 ; Stop to Start minimum time = 140 usecs
000000FA          00126 TLT_MAX equ .250/PrSclr2 ; Stop to Start maximum time = 260 usecs
000000B4          00127 TLT_MID equ .180/PrSclr2 ; Stop to Start median time  = 208 usecs
0000004A          00128 SRQ_MAX equ .296/PrSclr1 ; amount of time to hold for a Service ReQuest
                  00129
                  00130 ;NOTE: for Timer0 timing of sending bits, some extra time is allowed for
                  00131 ;instruction cycles between the end of the bit and the start of the next
                  00132 ; bit
00000016          00133 LOW1BIT equ .22/PrSclr2  ; low time for a 1 bit
00000032          00134 HI_1BIT equ .50/PrSclr2  ; hi time  for a 1 bit
```


© 1997 Microchip Technology Inc.

```
  00000038        00135 LOW0BIT equ  .56/PrSclr2  ; low time for a 0 bit
  00000014        00136 HI_0BIT equ  .20/PrSclr2  ; hi time  for a 0 bit
                  00137
                  00138
                  00139 ;*** ADB DATA REGISTERS - 2 BYES FOR EACH OF REGISTERS 0, 1, 2, and 3
                  00140
0008              00141 ADB_REG ORG   08h     ; ORIGIN FOR ADB DATA REGISTERS
0008              00142 Reg0a   RES   01h     ; 8
0009              00143 Reg0b   RES   01h     ; 9
000A              00144 Reg1a   RES   01h     ; A
000B              00145 Reg1b   RES   01h     ; B
000C              00146 Reg2a   RES   01h     ; C
000D              00147 Reg2b   RES   01h     ; D
000E              00148 Reg3a   RES   01h     ; E
000F              00149 Reg3b   RES   01h     ; F
                  00150
                  00151
                  00152 ;* VARIABLE REGISTERS FOR STORAGE, FLAGS, THE TIME VARIABLE,
                  00153 ;  THE COUNTER, & RANDOM VALUES
                  00154
0010              00155 STORAGE ORG   10h  ; ORIGIN FOR MISC. DATA VARIABLES
0010              00156 TmpReg1 RES   01h  ; 10 - temporary registers where Data is sent from &
0011              00157 TmpReg2 RES   01h  ; 11 -  received; NOTE: THESE 2 MUST BE IN THIS ORDER
0012              00158 RegNum  RES   01h  ; 12 - holds current ADB Data Reg.#-NOT a RAM address
0013              00159 RAMaddr RES   01h  ; 13 - holds RAM address of ADB Data Reg.#
0014              00160 Flags1  RES   01h  ; 14 - two Flags registers used by ADB & 2nd
0015              00161 Flags2  RES   01h  ; 15 -   Application Task
0016              00162 CmdByte RES   01h  ; 16 - holds the Command Byte
0017              00163 BitCntr RES   01h  ; 17 - counts down when sending or receiving bits
0018              00164 Random  RES   01h  ; 18 - stores Random Address sent in Talk routine
0019              00165 TimeVar RES   01h  ; 19 - used with TMR0 for all ADB timing
001A              00166 Tsk2Var RES   01h  ; 1A - used with TMR0 for timing during 2nd Task
                  00167
                  00168
                  00169 ;*** REGISTERS STILL AVAILABLE
                  00170
001B              00171 TmpCtr1 RES   01h  ; 1B
001C              00172 TmpFlg1 RES   01h  ; 1C
001D              00173 TmpFlg2 RES   01h  ; 1C
001E              00174 TmpFlg3 RES   01h  ; 1D
001F              00175 TmpFlg4 RES   01h  ; 1E
                  00176
                  00177
0000              00178 PROGRAM ORG     00h     ; origin for program
                  00248     include "adb.sub"   ; ADB Sub-Routines - these must be included
                  00001 ;*************************************************************************
                  00002 ;*************************************************************************
                  00003 ;   ******           THE FOLLOWING ARE SUB-ROUTINES            *******
                  00004 ;   ******             CALLED BY THE MAIN PROGRAM              *******
                  00005 ;*************************************************************************
                  00006 ;*************************************************************************
                  00007
                  00008 ;*** SWITCH PRESCALER BETWEEN WDT AND Timer0 *** (PrScale, NoPrScl) ***
                  00009 ;*** THIS PROCEDURE, DOCUMENTED IN SPEC. SHEET SECTION 9.1, IS INTENDED
                  00010 ;*** TO PREVENT UNEXPECTED RESET CONDITION
                  00011
                  00012 ;*** PrScale ROUTINE CALLED AT END OF AttnSig AND Srq SIGNALS
0000 0004         00013 PrScale clrwdt              ; Change prescaler from WDT to TMR0
0001 0C01         00014         movlw   b'00000001' ; BINARY - set to prescale TMR0
0002 0002         00015         option              ; Clear 4th bit from right to select TMR0
0003 0061         00016         clrf    TMR0        ; last 3 bits set prescale value as 1:4
0004 0800         00017         retlw   NULL        ;  this gives a good ratio to monitor the
                  00018                             ;  timing for Reset and Attention signals and
                  00019                             ;  the 2nd Application Task
                  00020
                  00021 ;***NoPrScl ROUTINE CALLED AT BEGINNING OF SyncSig AND END OF Srq SIGNALS
```


© 1997 Microchip Technology Inc.

```
0005 0061          00022 NoPrScl clrf    TMR0        ; Change prescaler from TMR0 to WDT
0006 0C08          00023         movlw   b'00001000' ; Set 4th bit from right to select WDT
0007 0002          00024         option
0008 0004          00025         clrwdt
0009 0C08          00026         movlw   b'00001000'
000A 0002          00027         option
000B 0800          00028         retlw   NULL
                   00029
                   00030 ;*************************************************************************
                   00031
                   00032 ;*GET INCOMING BIT & INTERPRET WHETHER IT'S A '1' OR A '0' *** (Get_Bit)*
                   00033 ;*** Get_Bit CALLED BY COMMAND AND LISTEN ROUTINES
                   00034 ; Get the bit, find out whether it's less than or greater than 50 usecs,
                   00035 ; if < than 50 usecs, it's a '1' bit
                   00036 ; if > than 50 usecs, it's a '0' bit
                   00037 ; if it's a '1' bit, set LSB in the reg. pointed to by the FSR (Command
                   00038 ; Byte) if it's a '0' bit, do nothing to the LSB
                   00039 ; then look for the end of the Bit Cell (104 usecs max.)
                   00040 ; if the maximum Bit time of (72 usecs) or maximum Bit Cell time is
                   00041 ; exceeded, abort to the Attn Signal
                   00042
000C 0201          00043 Get_Bit movf   TMR0,W ; Check the time, then check if the line went high:
000D 0099          00044         subwf  TimeVar,W ; See if more than BIT_TST usecs have passed
000E 0703          00045         btfss  STATUS,C  ; if not, check whether the line went high
000F 0AAB          00046         goto   AttnSig   ; if so, abort to the Attn Signal
0010 0705          00047         btfss  PORTA,ADB ; Check whether the line went high
0011 0A0C          00048         goto   Get_Bit   ; if line is still low, loop again
0012 0C32          00049         movlw  BIT_TST   ; if line went high, see if it's a '1' or a '0'
0013 0039          00050         movwf  TimeVar   ; as the bit has not yet been determined yet,
0014 0400          00051         bcf    INDF,LSB  ; ensure the LSB in the indirect address is '0'
0015 0201          00052         movf   TMR0,W    ; Get the time
0016 0099          00053         subwf  TimeVar,W ;  if time < 50 usecs, it's a '1' bit
0017 0603          00054         btfsc  STATUS,C  ; if time > 50 usecs and < 72, it's a '0' bit
0018 0500          00055         bsf    INDF,LSB  ; if it's a 1, set LSB in the address FSR points
0019 0C68          00056         movlw  BITCELL   ; to Check whether the Max. Bit Cell time of
001A 0039          00057         movwf  TimeVar   ; 104 usecs has been exceeded
001B 0201          00058 CellChk movf   TMR0,W    ; Check the time, then check the line
001C 0099          00059         subwf  TimeVar,W ; See if more than Max. Bit Cell usecs have
001D 0703          00060         btfss  STATUS,C; passed if not, look for the line to go low again
001E 0AAB          00061         goto   AttnSig   ; if so, abort to the Attn Signal or Reset
001F 0605          00062         btfsc  PORTA,ADB ; Check the line for the start of another bit
0020 0A1B          00063         goto   CellChk   ; if the line is still high, loop CelChk1 again
0021 0061          00064         clrf   TMR0      ; if the line went low, clear the TMR0 & return
0022 0800          00065         retlw  NULL      ; for another bit or to interpret the Command
                   00066
                   00067 ;*************************************************************************
                   00068 ;* DETERMINE IF THIS IS A GLOBAL COMMAND TO ALL DEVICES  *** (Globals) *
                   00069 ;*** Globals CALLED BY AddrChK
                   00070
0023 0211          00071 Globals movf   TmpReg2,W ; Check whether the Command is for all devices
0024 0F04          00072         xorlw  C_RES_1   ; retrieve the Command Type (the upper 2 bits
0025 0643          00073         btfsc  STATUS,Z  ; of the Command nibble)
0026 0BC4          00074         goto   Reserv1   ; test for this being the first Reserved
0027 0210          00075         movf   TmpReg1,W ; Command retrieve the whole Command Nibble
0028 0F02          00076         xorlw  C_RES_2 ; test for this being the second Reserved Command
0029 0643          00077         btfsc  STATUS,Z
002A 0BC5          00078         goto   Reserv2
002B 0210          00079         movf   TmpReg1,W ; retrieve the whole Command Nibble
002C 0F03          00080         xorlw  C_RES_3   ; test for this being the third Reserved Command
002D 0643          00081         btfsc  STATUS,Z
002E 0BC6          00082         goto   Reserv3
002F 0F00          00083         xorlw  C_RESET   ; test for this being Reset Command
0030 0643          00084         btfsc  STATUS,Z
0031 0A96          00085         goto   Reset
0032 0800          00086         retlw  NULL
                   00087
```

```
                00088 ;*************************************************************************
                00089
                00090 ;* MASK OUT COMMAND NIBBLE AND REG.# BITS FROM THE COMMAND *** (MaskCmd)*
                00091 ; NOTE: This routine should only be called once during any single ADB
                00092 ; transaction, from either AddrChk or CmmdChk
                00093
0033 0216       00094 MaskCmd movf  CmdByte,W ; Mask the Command to save the Data Reg. # bits &
0034 0E0F       00095         andlw CMDNIBL   ; the Command Type bits (Listen, Talk, etc.):
0035 0030       00096         movwf TmpReg1   ; save the Command nibble
0036 0E0C       00097         andlw CMDTYPE   ; mask the upper 2 Command Type bits (Talk, etc.)
0037 0031       00098         movwf TmpReg2   ; save the upper 2 Command Type bits
0038 0216       00099         movf  CmdByte,W : extract the Data Register number:
0039 0E03       00100         andlw REGMASK   ; mask out Data Reg. number from Command Nibble
003A 0032       00101         movwf RegNum    ; save the Data Reg. bits
003B 0024       00102         movwf SR        ; save pointer to Data Reg. in File Select Reg.
                00103                         ; in order to setup RAM address where start
                00104                         ; of Data for this Reg. will be stored
003C 0403       00105 SaveRAM bcf     STATUS,C; clear Carry bit so it doesn't wrap around
003D 0364       00106         rlf     FSR,F   ; multiply by 2 to get 1st Byte of RAM addr
003E 0564       00107         bsf     FSR,03h ; add array offset for Send/Receive/Flush Reg.
003F 0204       00108         movf    FSR,W   ; by setting bit of 1st RAM address, which
0040 0E1F       00109         andlw   FSRMASK ; is ORG'd in ADB.EQU equates
0041 0033       00110         movwf   RAMaddr ; mask out the RAM address of Data Reg. Number
0042 0800       00111         retlw   NULL    ; save RAM address of Data Reg. and return
                00112
                00113 ;*************************************************************************
                00114
                00115 ;*** ISSUE A SERVICE REQUEST IF NECESSARY *** (Srq; may call LineLow) ***
                00116 ;*** CALLED BY AddrChk
                00117 ; see if the Srq Flag is set, if not, return, otherwise:
                00118 ; change the prescaler to TMR0 since this takes longer than 255 usecs,
                00119 ; load the SRQTIME of 300 usecs into the TimeVariable,
                00120 ; call LineLow to:
                00121 ; keep checking the time to see if 300 usecs have passed,
                00122 ; let the line go high again,
                00123 ; and see if the line is high, and if not, abort, if it is,
                00124 ; change the prescaler back to WDT, and return
                00125
0043 0715       00126 Srq     btfss   Flags2,F2Srq ; see if the Srq flag is set,
0044 0800       00127         retlw   NULL         ; if not, return
0045 0900       00128         call    PrScale      ; switch the prescaler to TMR0
0046 0C00       00129         movlw   TRI_OUT      ; tri-state PORTA to make the ADB an output
0047 0005       00130         tris    PORTA
0048 0C4A       00131         movlw   SRQ_MAX
0049 0976       00132         call    LineLow
004A 0905       00133         call    NoPrScl      ; change the prescaler back to WDT
004B 0800       00134         retlw   NULL
                00135
                00136 ;*************************************************************************
                00137
                00138 ;*** Tlt - TIME FROM STOP BIT TO START BIT ***  (Tlt) ***
                00139 ;*** CALLED BY EITHER Talk OR Listen ROUTINES
                00140 ; Loop checking the time, then checking the line to see if it went low
                00141 ; if at any time the line goes low,
                00142 ; see if this is a Talk Command,
                00143 ; if it is a Talk Commmand, go to the Collision routine
                00144 ; if the line goes low before the minimum Tlt time, abort to Attn Signal
                00145 ; if the line is high longer than TLT_Min usecs,
                00146 ; see if this is a Talk Command, and if it is, wait for the mid-point,
                00147 ; and return to Send the Start Bit, Data Bytes, & the Stop Bit
                00148 ; if it's not a Talk Command, see if it's a Listen Command, and if so,
                00149 ; load Tlt_Max for TimeVariable, and look for the line to go
                00150 ; low as the beginning of the Start Bit,
                00151 ; if more than Tlt_Max usecs pass, abort to Attn Signal
                00152 ; if the line goes low and this is a Listen Command,
                00153 ; clear the TMR0 & return to get the rest of the Start Bit
```


© 1997 Microchip Technology Inc.

```
                00154
004C 0C8C       00155 Tlt      movlw   TLT_MIN       ; Look for Stop-to-Start-Time, Tlt
004D 0039       00156          movwf   TimeVar       ; Check the time, then check the line
004E 0201       00157 TltChk1  movf    TMR0,W        ; See if more than TLT_MIN usecs have passed
004F 01B8       00158          xorwf   Random,F      ; (ensure the Talk R3 address is Random with
0050 0099       00159          subwf   TimeVar,W     ; XOR) by checking whether Carry bit is set
0051 0703       00160          btfss   STATUS,C      ; after subtraction
0052 0A5D       00161          goto    ChkFlag       ; if TLT_MIN usecs passed, see what Command
0053 0605       00162          btfsc   PORTA,ADB     ; this is if not, check whether the line went
0054 0A4E       00163          goto    TltChk1       ; low if the line is still high, keep looping
0055 0654       00164          btfsc   Flags1,F1Talk;  if line went low, see if this is a Talk
0056 0B5A       00165          goto    Collisn       ; Command if it is, there was a Collision,
0057 0201       00166          movf    TMR0,W        ; abort otherwise, check the time
0058 0099       00167          subwf   TimeVar,W     ; see if TLT_MIN usecs passed,
0059 0703       00168          btfss   STATUS,C      ; if not, abort to Attn Signal, too little
005A 0AAB       00169          goto    AttnSig       ; time passed when the line went low
005B 0061       00170          clrf    TMR0          ; if it's not a Talk Command, clear the TMR0
005C 0800       00171          retlw   NULL          ; and return for the rest of the Start Bit
                00172
005D 0654       00173 ChkFlag  btfsc   Flags1,F1Talk ; Check whether to Talk or Listen
005E 0A6D       00174          goto    TltTalk       ; if Talk, wait for mid-point of Tlt time
005F 0794       00175          btfss   Flags1,F1Lstn ; if Listen, continue to look for Start Bit
0060 0800       00176          retlw   NULL          ; if neither flag is set, abort, something's
0061 0CFA       00177          movlw   TLT_MAX       ; wrong  Load TimeVariable to check for
0062 0039       00178          movwf   TimeVar       ; upper limit of Tlt time
0063 0201       00179 TltChk2  movf    TMR0,W        ; See if TLT_MAX usecs have been exceeded
0064 0099       00180          subwf   TimeVar,W     ; by checking whether Carry bit is set
0065 0703       00181          btfss   STATUS,C      ; after subtraction
0066 0AAB       00182          goto    AttnSig       ; if so, abort to Attn Signal
0067 0605       00183          btfsc   PORTA,ADB     ; if not, check whether the line went low
0068 0A63       00184          goto    TltChk2       ;if line is still high, check the time again
0069 0654       00185          btfsc   Flags1,F1Talk ;if line went low, see if this is a Talk
006A 0B5A       00186          goto    Collisn       ; Command if so, there was a Collision
006B 0061       00187          clrf    TMR0          ; if it's not a Talk Command, return to get
006C 0800       00188          retlw   NULL          ; the rest of the Start Bit from Host
                00189
006D 0CB4       00190 TltTalk  movlw   TLT_MID       ; Load TimeVariable so Talk will send Start
006E 0039       00191          movwf   TimeVar       ; Bit at about the mid-point of the Tlt
006F 0201       00192 TltChk3  movf    TMR0,W        ; See if TLT_MID usecs have been exceeded
0070 0099       00193          subwf   TimeVar,W     ; by checking whether Carry bit is set
0071 0703       00194          btfss   STATUS,C      ; after subtraction
0072 0800       00195          retlw   NULL        ; if time was exceeded, return to send Start Bit
0073 0605       00196          btfsc   PORTA,ADB     ; if not, check whether the line went low
0074 0A6F       00197          goto    TltChk3       ; if line is still high, check the time again
0075 0B5A       00198          goto    Collisn       ; if the line went low, abort to Collision
                00199
                00200 ;*************************************************************************
                00201
                00202 ;*** MAKE LINE GO LOW TIME IN TimeVar AS A '1' OR '0' BIT***(LineLow)***
                00203 ;*** CALLED BY Talk OR Srq
                00204
0076 0039       00205 LineLow  movwf   TimeVar     ;
0077 0201       00206 Low_Tmp  movf    TMR0,W        ; Check the clock,
0078 0099       00207          subwf   TimeVar,W     ; loop until TimeVar usecs have passed
0079 0603       00208          btfsc   STATUS,C;
007A 0A77       00209          goto    Low_Tmp       ;
007B 0C01       00210          movlw   TRI_IN        ; Tri-state PORTA to make ADB line an input
007C 0005       00211          tris    PORTA         ; again and let the line go high
007D 0061       00212          clrf    TMR0          ; and clear TMR0
007E 0000       00213          nop                   ; Allow the ADB Port line to stabilize
007F 0000       00214          nop                   ; Allow the ADB Port line to stabilize
0080 0705       00215          btfss   PORTA,ADB     ; check if the line is still low, if so, a
0081 0B5A       00216          goto    Collisn       ; Collision occurred
0082 0800       00217          retlw   NULL    ; if not, return to load high time for rest of bit
                00218
                00219 ;* MAKE LINE GO HIGH FOR REST OF BIT CELL TIME IN TimeVar *** (LineHi)*
```

```
                        00220 ;*** CALLED BY Talk
                        00221
0083 0039               00222 LineHi  movwf   TimeVar  ; Let the line go high for a pre-designated time
0084 0201               00223 Hi_Tmp  movf    TMR0,W   ; Check the clock,
0085 0099               00224         subwf   TimeVar,W; loop until TimeVar usecs have passed
0086 0603               00225         btfsc   STATUS,C ;
0087 0A84               00226         goto    Hi_Tmp   ;
0088 0705               00227         btfss   PORTA,ADB; check if the line is still high,
0089 0B5A               00228         goto    Collisn  ; if not, a Collision occurred, Abort
008A 0674               00229         btfsc   Flags1,F1Stop; if this is the end of the Data Stop Bit,
008B 0800               00230         retlw   NULL     ; don't let the line go low again, just return
008C 0C00               00231         movlw   TRI_OUT  ; if still high, start sending a bit to the Host
008D 0005               00232         tris    PORTA    ; tri-state PORTA to make the ADB an output and
008E 0061               00233         clrf    TMR0     ; return
008F 0800               00234         retlw   NULL     ;
                        00235
                        00236 ;*************************************************************************
                        00237 ;*************************************************************************
                        00238 ;   ******                  END OF SUB-ROUTINES                 *******
                        00239 ;*************************************************************************
                        00240 ;*************************************************************************
                        00249                           ; here to ensure being in the first
                        00250                           ; half of the memory page when called.
                        00251
                        00252 IntData macro DataCmd,Routine ; Macro goes to an appropriate Listen Reg.3
                        00253         movf    TmpReg2,W ; interprets the Data Command received by
                        00254         xorlw   DataCmd   ; comparing the 2nd byte to a Data
                        00255         btfsc   STATUS,Z  ; Command constant
                        00256         goto    Routine   ; it then goes to the appropriate routine
                        00257         endm
                        00258
                        00259 ;
                        00260 ;*** CONDITIONAL ASSEMBLY DETERMINED BY LIST DIRECTIVE
                        00261 ;
                        00262         ifdef      __16C56
                        00263         include "5657mcro.mod"     ; macros for the 2nd Application Task
                        00001 ;
                        00002 ;*** LoadEm MACRO USED FOR TESTING DURING 2ND APPLICATION TASK
                        00003 ;*** ONLY FOR PART 16c56/57
                        00004 ;
00000004                00005 H       equ     04h     ; *** THESE ARE USED AS KEYS PRESSED WHEN PART
0000000E                00006 E       equ     0Eh     ; ***   IS SELECTED FOR 16C56/57
00000025                00007 L       equ     25h
0000001F                00008 O       equ     1Fh
00000031                00009 SP      equ     31h
0000000D                00010 WW      equ     0Dh     ; W is already defined in the PICREG5X.EQU file
0000000F                00011 R       equ     0Fh
00000002                00012 D       equ     02h
00000024                00013 RETRN   equ     24h
000000FF                00014 FILLCHR equ     0FFh    ; 'fill character' as described in spec.
                        00015 ;
                        00016 ;
                        00017 LoadEm  macro   Ctr,Bit,Dest,RegA,RegB; Macro used to load registers and
                        00018         btfss   Ctr,Bit       ;  set  flags for Key-Up Transition Codes
                        00019         goto    Dest          ; Bits are cleared as the data is sent
                        00020         movlw   Reg0a
                        00021         movwf   FSR
                        00022         movlw   RegA          ; load data to be sent from register A
                        00023         movwf   INADDR
                        00024         incf    FSR,F
                        00025         movlw   RegB          ; load data to be sent from register B
                        00026         movwf   INADDR        ; load data to be sent from register B
                        00027         bsf     Flags2,F2DSend; Data now needs to be sent to the host
                        00028         bsf     Flags2,F2Srq  ; Until all data has been sent, Srq's may
                        00029         btfsc   Flags2,F2STest; be sent. See if Key Transition Codes
                        00030         goto    KeyUp         ; should be sent if so, go set the bits
```


© 1997 Microchip Technology Inc.

```
                00031       bsf     Flags2,F2STest; if not, set bits so they'll be next time
                00032       bcf     Ctr,Bit       ; clear the bit so next data will be sent
                00033       goto    DBounce       ; and go debounce the switch
                00034       endm
                00035
                00036
                00037
                00264       endif                 ;   the program is for a 16C56
                00265       ifdef      __16C57    ;   or 16C57 part
                00266       include "5657mcro.mod"
                00267       endif
                00268
                00269 ;*****************************************************************
                00270 ;*****************************************************************
                00271 ;               THE MAIN PROGRAM STARTS BELOW
                00272 ;*****************************************************************
                00273 ;*****************************************************************
                00274
0090 0C01       00275 Start movlw   TRI_IN    ; Start off by making the ADB pin an
0091 0005       00276       tris    PORTA     ; input on PORTA
0092 0405       00277       bcf     PORTA,ADB ; make line will go low when tris'd as an output
                00278
                00279 ; *** THIS I/O SETUP ROUTINE IS USED FOR TESTING WITH AN LED ON RB0
                00280 ; *** AND A SWITCH ON RB1
0093 0C02       00281 TSTING1 movlw   b'00000010'; Make RB0 an output (for the LED) and
0094 0006       00282       tris    PORTB     ; RB1 an input (for the normally open switch)
0095 0406       00283       bcf     PORTB,LED ; Make sure the LED is off to begin with
                00284
                00285 ;*****************************************************************
                00286
0096 0705       00287 Reset btfss   PORTA,ADB ; Reset Signal - loop until the line is high,
0097 0A96       00288       goto    Reset     ; then initialize Registers
                00289
0098 0070       00290 Init  clrf    TmpReg1   ; Initialization routine
0099 0071       00291       clrf    TmpReg2   ; Clear variables
009A 0072       00292       clrf    RegNum    ; NOTE: No need to clear variable register
009B 0073       00293       clrf    RAMaddr   ;  'Random' as it is XOR'd in other routines
009C 0074       00294       clrf    Flags1    ;  to produce a random Address for the 'Talk
009D 0075       00295       clrf    Flags2    ;  Reg. 3' Command
009E 0077       00296       clrf    BitCntr
009F 0068       00297       clrf    Reg0a     ; Clear ADB Storage Data Register Variables
00A0 0069       00298       clrf    Reg0b
00A1 006A       00299       clrf    Reg1a
00A2 006B       00300       clrf    Reg1b
00A3 006C       00301       clrf    Reg2a
00A4 006D       00302       clrf    Reg2b
                00303
00A5 0C02       00304       movlw   DEF_ADD   ; Register 3 has special Default Data set at
00A6 002E       00305       movwf   Reg3a     ; Reset: load Register 3a with Default Device
00A7 05AE       00306       bsf     Reg3a,Srq_Bit; Address allow Service Requests of Host
00A8 05CE       00307       bsf     Reg3a,ExpEvnt;  include the Exceptional Event bit as
                00308                       ;default * NOTE: at this time, this Device
                00309                       ; doesn't process for Exceptional Events
00A9 0C03       00310       movlw   DEF_HND   ;
00AA 002F       00311       movwf   Reg3b     ; load Register 3b with Default Device Handler ID
                00312
                00313 ;*****************************************************************
                00314
                00315 ;*** LOOK FOR ATTENTION OR RESET ***  (AttnSig)  ***
                00316 ; Look for the line being low, when it is, see if the line went high.
                00317 ; During that time, allow the 2nd Application Task to be performed for a
                00318 ; limited amount of time, then return to Attn Signal
                00319 ; if the line went high, did it go high within the 776-824 usec range?
                00320 ; if so, go on to get the Command
                00321 ; if not, goto the Reset routine
                00322 ; IN DETAIL:
```


© 1997 Microchip Technology Inc.

```
                00323 ;   look at the line
                00324 ;   if the line is not yet low,
                00325 ;   loop until it goes low, & clear the TMR0
                00326 ;   Loop with Minimum Time: check the time
                00327 ;   if the time is less than the Attention Minimum usecs,
                00328 ;   check whether the line has gone high,
                00329 ;   if the line has not gone high,
                00330 ;   loop again checking the time
                00331 ;   if the line has gone high,
                00332 ;   check whether the Min. usecs have passed
                00333 ;   if not, Abort; too little time went by.
                00334 ;   if so, go on to look for the Sync signal
                00335 ;   Loop with Maximum Time: load the Maximum Time Variable & check
                00336 ;   the time if the time is less than the Attention Maximum usecs,
                00337 ;   check whether the line has gone high,
                00338 ;   if the line has not gone high,
                00339 ;   loop again checking the time
                00340 ;   if the line has gone high before Max. Attention usecs have passed,
                00341 ;   go on to look for the Sync signal
                00342 ;   if the time is greater than the Attention Maximum usecs,
                00343 ;   abort to Reset
                00344
                00345 ;*****************************************************************************
                00346
00AB 0201       00347 AttnSig movf   TMR0,W        ; Look for Attn between ATT_MIN - ATT_MAX usecs
00AC 07F4       00348         btfss  Flags1,F1Cllsn; this is a good time to use the TMR0 and
00AD 01B8       00349         xorwf  Random,F      ; Pseudo-Random Address
00AE 0605       00350         btfsc  PORTA,ADB     ; See if the line went low
00AF 0AAB       00351         goto   AttnSig       ; Loop to AttnSig until the line goes low
00B0 0900       00352         call   PrScale       ; Switch prescaler to TMR0 for > 250 usec count
                00353                              ; during Attn Signal
00B1 0CC2       00354         movlw  ATT_MIN       ;
00B2 0039       00355         movwf  TimeVar       ; use TimeVariable to subtract from ATT_MIN usecs
                00356
00B3 0076       00357 CleanUp clrf   CmdByte       ; Clear the Command Byte
00B4 0070       00358         clrf   TmpReg1       ; Clear the temporary Data registers
00B5 0071       00359         clrf   TmpReg2       ; NOTE: No need to clear variable register
00B6 0072       00360         clrf   RegNum        ; 'Random' clear the current Register Number
00B7 0073       00361         clrf   RAMaddr       ; register clear the register holding the RAM
                00362                              ; Address of the 1st byte of where Data is stored
00B8 0514       00363         bsf    Flags1,F1Attn; Set this bit to indicate to the 2nd Task
                00364                              ; that it should Return to the AttnMin routine
00B9 0434       00365         bcf    Flags1,F1Reg3    ; Clear Flags: Data-for-Register 3
00BA 0454       00366         bcf    Flags1,F1Talk    ; Talk
00BB 0474       00367         bcf    Flags1,F1Stop    ; Data-Stop-Bit-is-being-sent
00BC 0494       00368         bcf    Flags1,F1Lstn    ; Listen
00BD 04B4       00369         bcf    Flags1,F1Sent1   ; Sent-1st-Byte
00BE 04D4       00370         bcf    Flags1,F1Rcvd1   ; Received-1st-Byte
                00371
00BF 0C7D       00372         movlw  TSK2MIN       ; load Task 2 Time Variable with amount allowed
00C0 003A       00373         movwf  Tsk2Var       ; during Attn Signal
00C1 0BCB       00374         goto   Task_2        ; This space allows running a second application
                00375                              ; NOTE: BE SURE TO RETURN TO ATTNMIN BEFORE 750
                00376                              ; usecs HAVE PASSED, AND DON'T LET THE OTHER
                00377                              ; APPLICATION AFFECT THE Timer0 or TimeVar.
                00378
00C2 0201       00379 AttnMin movf   TMR0,W        ; Check the time, then check the line
00C3 0099       00380         subwf  TimeVar,W     ; See if more than ATT_MIN usecs have passed
00C4 0703       00381         btfss  STATUS,C      ; if not, check the line
00C5 0ACD       00382         goto   AttnMax       ; if so, go check time/line again in AttnMax
00C6 0705       00383         btfss  PORTA,ADB     ; Check for line being high & if so, check time
00C7 0AC2       00384         goto   AttnMin       ; if line is still low, loop again
00C8 0201       00385         movf   TMR0,W        ; if line is high, see if time is in range
00C9 0099       00386         subwf  TimeVar,W     ; by checking whether Carry bit is
00CA 0703       00387         btfss  STATUS,C      ; set after subtraction
00CB 0AAB       00388         goto   AttnSig       ; If time <= Min, look for Attn Signal again
```


© 1997 Microchip Technology Inc.

```
00CC 0AD6          00389          goto    SyncSig    ; If time > Min, go get Sync signal
                   00390
00CD 0CCE          00391 AttnMax  movlw   ATT_MAX    ; Load the TimeVariable to check for the
00CE 0039          00392          movwf   TimeVar    ; maximum amount of time for Attn Signal
00CF 0201          00393 AttnTmp  movf    TMR0,W     ; Check the time, then check the line
00D0 0099          00394          subwf   TimeVar,W  ; See if more than ATT_MAX usecs have passed
00D1 0703          00395          btfss   STATUS,C   ; if not, check the line
00D2 0A96          00396          goto    Reset      ; if so, Abort to Reset;too much time has passed
00D3 0705          00397          btfss   PORTA,ADB  ; Check for the line to going high
00D4 0ACF          00398          goto    AttnTmp    ; if the line isn't high, loop AttnMax again
00D5 0061          00399          clrf    TMR0       ; if the went high, go get the Sync signal
                   00400
                   00401 ;*************************************************************************
                   00402
                   00403 ;*** LOOK FOR SYNC SIGNAL ***  (SyncSig) ***
                   00404 ; This routine checks the timing between the rising edge of the Attention
                   00405 ; Signal & a falling edge indicating the start of the 1st Command bit.
                   00406 ; At the end of the Attn Signal routine, the line went high, and
                   00407 ; the TMR0 was cleared.
                   00408 ; Check the TMR0,
                   00409 ; if the 72 usec limit is exceeded,
                   00410 ; abort to the Attn Signal
                   00411 ; if the 72 usec limit is not exceed,
                   00412 ; check the line
                   00413 ; if the line went low (as the first bit of the Command),
                   00414 ; go on to get the 8 Command Bits
                   00415 ; if the line is still high,
                   00416 ; loop to check TMR0 again
                   00417
                   00418 ;*************************************************************************
                   00419
00D6 0905          00420 SyncSig  call    NoPrScl    ; Get the Sync Signal which follows the Attn
00D7 0C48          00421          movlw   SYNC       ; Signal Turn off prescaler; timing counts are
00D8 0039          00422          movwf   TimeVar    ; < 255 usecs  and load the timing the for the
00D9 0099          00423 SyncTmp  subwf   TimeVar,W  ; Sync Signal See if more than SYNC usecs
00DA 0703          00424          btfss   STATUS,C   ; have passed if not, go check the line
00DB 0AAB          00425          goto    AttnSig    ; if so, Abort to Attn Signal
00DC 0605          00426          btfsc   PORTA,ADB  ; Check for the line to go low
00DD 0AD9          00427          goto    SyncTmp    ; if the line is still high, loop again
00DE 0061          00428          clrf    TMR0       ; if low, clear TMR0 & go on to get the Command
                   00429
                   00430 ;*************************************************************************
                   00431
                   00432 ;*** GET THE COMMAND: 8 BITS & STOP BIT ***  (Command) ***
                   00433 ; The Sync Signal was detected when the line went low after approximately
                   00434 ; 70 usecs.  This low line is the first bit of the Command.  This
                   00435 ; routine receives 8 bits, followed by a '1' Stop bit.
                   00436
                   00437 ; IN DETAIL:
                   00438 ; initialize a counter for counting down as the bits come in
                   00439 ; call Get_Bit to receive each bit, MSB first, & rotate it into the
                   00440 ; CmdByte register, where the Command Byte is stored.
                   00441 ; After returning from GetBit, decrement the counter.
                   00442 ; when all 8 bits have been received, clear TMR0 (to allow looking
                   00443 ; for the Stop bit, or holding down the line for an SRQ), and go on to
                   00444 ; Interpret the Command.
                   00445
                   00446 ; In GetBit, get the time,
                   00447 ; if the time is greater than 72 usecs,
                   00448 ; abort to the Attn Signal
                   00449 ; if the time is less than 72 usecs,
                   00450 ; check if the line went high
                   00451 ; if line is still low,
                   00452 ; loop to check the time again
                   00453 ; if the line went high,
                   00454 ; determine whether the line went high before or after 50 usecs
```


© 1997 Microchip Technology Inc.

```
                00455 ; if the line went high before 50 usecs, rotate a 1 bit into CmdByte reg.
                00456 ; if the line went high after 50 usecs, rotate a 0 bit into CmdByte reg.
                00457 ; after getting a bit, check if the line went low (the start of the next
                00458 ; bit) if the max. Cell Bit time (104 usecs) is exceeded, abort to Attn
                00459 ; Signal when the line goes low, clear TMR0 and return to get another
                00460 ;  bit or  interpret the Command if all 8 bits have been been received
                00461
                00462 ;*****************************************************************************
                00463
00DF 0C08       00464 Command movlw   BYTE      ; Get the 8 Command Bits - 1st bit already
00E0 0037       00465         movwf   BitCntr   ; started, so count down from 8 to 0
00E1 0C16       00466         movlw   CmdByte   ; rotate bits into CmdByte with indirect
00E2 0024       00467         movwf   FSR       ; address
00E3 0C48       00468 CmdLoop movlw   MAX_BIT   ; Get & rotate a 1 or 0 bit into CmdByte, or
00E4 0039       00469         movwf   TimeVar   ; see if the maximum time is exceeded & abort
00E5 0403       00470         bcf     STATUS,C  ; clear Carry bit to ensure it won't wrap around
00E6 0376       00471         rlf     CmdByte,F ; rotate in the last bit
00E7 090C       00472         call    Get_Bit   ; and get another one
00E8 02F7       00473         decfsz  BitCntr,F ; keep looping until 8 bits are received &
00E9 0AE3       00474         goto    CmdLoop   ; rotated when the Command has been received,
                00475                           ; interpret it
                00476 ;*****************************************************************************
                00477
                00478 ;*** CHECK THE ADDRESS ***  (AddrChk; may call MaskCmd, Globals, Srq) ***
                00479 ; The Command Stop Bit is a good time to determine if the Host is
                00480 ; addressing this Device:
                00481 ; test the left nibble of the received byte against the current Address
                00482 ; if the Address belongs to this Device,
                00483 ; mask out the command and register nibble of the received byte,
                00484 ; test it to see whether the Command is to Listen, Talk, or Flush
                00485 ; and go to the routine that looks for the end of the Stop Bit
                00486 ; if the Command is for another Device,
                00487 ; mask the command nibble
                00488 ; see if the Command is a global/reserved Command
                00489 ; if so, go do the Command
                00490 ; if the Command is not global,
                00491 ; check the Srq flag to see if another application needs service
                00492 ; if the Srq flag is set,
                00493 ; go issue a Service Request (Srq)
                00494 ; if the Srq flag is not set,
                00495 ; go get the Attn Signal
                00496
00EA 020E       00497 AddrChk movf    Reg3a,W   ; See if the Command received is for this Device
00EB 0E0F       00498         andlw   DEVMASK   ; by masking off this Device's Address
00EC 0031       00499         movwf   TmpReg2   ; and saving it in a temporary register
00ED 03B1       00500         swapf   TmpReg2,F ; (received nibbles in Command are reversed)
00EE 0216       00501         movf    CmdByte,W ; Test if the received Address is for Device,
00EF 0EF0       00502         andlw   ADDRMSK   ; by masking out the Command nibble,
00F0 0191       00503         xorwf   TmpReg2,W ; compare received Address to current Address
00F1 0643       00504         btfsc   STATUS,Z  ; if Address is for this Device, go get the Stop
00F2 0AF7       00505         goto    CmdStop   ; Bit & see what the Command is for this Device.
                00506
00F3 0933       00507         call    MaskCmd   ; Mask the Command Nibbles from the Address
00F4 0923       00508         call    Globals   ; and go see if it was a Global Command
00F5 0943       00509         call    Srq       ; if not, go see if Srq needs to be asserted
00F6 0AAB       00510         goto    AttnSig   ; if not, go get the Attn Signal
                00511
                00512 ;*****************************************************************************
                00513
                00514 ;*** LOOK FOR THE COMMAND STOP BIT *** (CmdStop) ***
                00515 ; Look for the Stop Bit following the Command Byte.  This is not executed
                00516 ; if Srq is asserted by this Device.
                00517
00F7 0C48       00518 CmdStop movlw   MAX_BIT   ; load the maximum time for a bit low time
00F8 0039       00519         movwf   TimeVar   ;
00F9 0099       00520         subwf   TimeVar,W ; See if more than the max. # of usecs have
```


© 1997 Microchip Technology Inc.

```
00FA 0703       00521         btfss   STATUS,C     ; passed if not, go check for the line to go
00FB 0AAB       00522         goto    AttnSig      ; high if so, abort to the Attn Signal
00FC 0705       00523         btfss   PORTA,ADB    ; Check for the line to go high
00FD 0AF7       00524         goto    CmdStop      ; if the line is still low, loop CmdStop
00FE 0061       00525         clrf    TMR0         ; again if high, clear TMR0 as the beginning
                00526                              ; of the Tlt and go on to interpret Command
                00527                              ; as Talk, Listen, or Flush.
                00528
                00529 ;*************************************************************************
                00530
                00531 ;*** INTERPRET THE COMMAND ***  (CmmdChk) ***
                00532 ; Determine first if the command is for Register 3, and set the Reg3 flag
                00533 ; if so, then see if the Command is to Talk, Listen, or Flush and go to
                00534 ; that routine.
                00535
00FF 0933       00536 CmmdChk call    MaskCmd      ; Separate the Command Nibbles into temp. regs.
0100 0712       00537         btfss   RegNum,00h   ; (MaskCmd put Command Type bits into TmpReg1)
0101 0B04       00538         goto    CmdChk2      ; see if the Command is for Register 3
0102 0632       00539         btfsc   RegNum,01h   ; if not, go continue interpreting the Command
0103 0534       00540         bsf     Flags1,F1Reg3; if so, set the Reg. 3 flag indicating this
                00541                              ; condition for the Talk or Listen routines
                00542
0104 0211       00543 CmdChk2 movf    TmpReg2,W    ; Test what Command was received &
0105 0F0C       00544         xorlw   C_TALK       ;   branch accordingly
0106 0643       00545         btfsc   STATUS,Z     ; test for this being a Talk Command
0107 0B11       00546         goto    Talk
0108 0211       00547         movf    TmpReg2,W
0109 0F08       00548         xorlw   C_LISTN
010A 0643       00549         btfsc   STATUS,Z     ; test for this being a Listen Command
010B 0B5D       00550         goto    Listen
010C 0211       00551         movf    TmpReg2,W
010D 0F01       00552         xorlw   C_FLUSH
010E 0643       00553         btfsc   STATUS,Z     ; test if the Command is to Flush a Register
010F 0BBE       00554         goto    Flush        ; if the Command isn't a Flush, go get
0110 0AAB       00555         goto    AttnSig      ;   the Attn Signal
                00556
                00557 ;*************************************************************************
                00558
                00559 ;*** SEND DATA TO THE HOST *** (Talk; calls Tlt, LineLow, LineHi) ***
                00560 ; Data is sent to Host from ADB Data Registers using indirect addressing.
                00561 ; (TMR0 was cleared in CmmdChk, and timing for Tlt began there)
                00562 ; Call the Tlt (Stop to Start Time), which waits for the middle of the
                00563 ; Tlt, when the Tlt returns, send a '1' Start Bit,
                00564 ; load the first byte of the Data Register into temporary register,
                00565 ; send the 1st 8 bits,
                00566 ; load the second byte of the Data Register into temporary register,
                00567 ; send the 2nd 8 bits,
                00568 ; and send a '0' Stop Bit
                00569 ; if at anytime during the Tlt, LineLow, or LineHi the ADB line is
                00570 ; inappropriately high or low, the routine aborts to the Collision
                00571 ; routine. The Collision routine only sets a flag if this is a Talk Reg.
                00572 ; 3 Command, indicating a Collision occurred when sending Data for Reg.
                00573 ; 3, and goes  to get the Attention Signal.
                00574 ; Using temporary registers ensures the Data doesn't get cleared until
                00575 ; all of it has been sent.
                00576
0111 0634       00577 Talk  btfsc  Flags1,F1Reg3  ; if the talk command is for Register 3,
0112 0B1D       00578       goto   SetRndm        ; go create  a Random Address and load it into
0113 07D5       00579       btfss  Flags2,F2DSend ; TmpReg1 Check whether there is data to
0114 0AAB       00580       goto   AttnSig        ; send if not, let the bus timeout & get Attn
                00581
0115 0213       00582 SetTmps movf  RAMaddr,W     ; Signal Load the temporary registers with Data
0116 0024       00583         movwf FSR           ; stored at the appropriate RAM Address for the
0117 0200       00584         movf  INDF,W        ; Register indicated in the Command Byte
0118 0030       00585         movwf TmpReg1
0119 02A4       00586         incf  FSR,F
```


© 1997 Microchip Technology Inc.

```
011A 0200    00587          movf  INDF,W          ;Load 2nd temporary register from 2nd RAM
011B 0031    00588          movwf TmpReg2         ;Address where Data is stored
011C 0B29    00589          goto  CallTlt
             00590
011D 0201    00591 SetRndm  movf  TMR0,W          ;The Address sent to the Host for a Talk Reg.3
011E 0198    00592          xorwf Random,W        ;Command must be random to avoid collisions
011F 0E0F    00593          andlw LOW_NBL         ;with other Device Addresses during
0120 0030    00594          movwf TmpReg1         ;initialization
0121 020E    00595          movf  Reg3a,W         ;
0122 0EF0    00596          andlw HI_NIBL         ;
0123 0130    00597          iorwf TmpReg1,F       ;
0124 0634    00598 SetHndl  btfsc Flags1,F1Reg3   ; if this is a Talk R3 Command,
0125 020F    00599          movf  Reg3b,W         ; send the Device Handler ID
0126 0695    00600         btfsc  Flags2,F2SFail  ; if a Device Self-Test was performed and it
0127 0040    00601          clrw                  ; failed, send the reserved Handler ID of
0128 0031    00602          movwf TmpReg2         ; '00h' to indicate the Failed condition
             00603
0129 0554    00604 CallTlt bsf   Flags1,F1Talk   ; Set the Talk Flag to indicate to the Tlt
012A 094C    00605          call  Tlt             ; routine to return for the end of Talk Start Bit
             00606
012B 0C10    00607 SndStrt  movlw TmpReg1         ; Send a '1' bit as the Start Bit
012C 0024    00608          movwf FSR             ; Use the indirect addressing of the temporary
012D 0C00    00609          movlw TRI_OUT         ; registers from which Data will be sent
012E 0005    00610          tris  PORTA           ; tri-state PORTA to make the ADB an output
012F 0061    00611          clrf  TMR0            ; clear TMR0 as the beginning of a bit
0130 0C16    00612          movlw LOW1BIT
0131 0976    00613          call  LineLow         ; hold the line low for 1/3rd of a Bit Cell
0132 0C32    00614          movlw HI_1BIT
0133 0983    00615          call  LineHi          ; let the go line high for rest of the Bit Cell
             00616
0134 0C08    00617 SetSend  movlw BYTE            ; Send the data bytes
0135 0037    00618          movwf BitCntr         ; Load the counter to send 8 Bits
0136 06E0    00619 SndBits  btfsc INDF,MSB        ; determine whether to complete the send of
0137 0B3D    00620          goto  Send1           ; a '1' or '0' bit
             00621
0138 0C38    00622 Send0    movlw LOW0BIT         ; Send a '0' bit
0139 0976    00623          call  LineLow         ; hold the line low for 2/3rd of a Bit Cell
013A 0C14    00624          movlw HI_0BIT
013B 0983    00625          call  LineHi          ; let the line high for the rest of the Bit Cell
013C 0B41    00626          goto  Rotate
             00627
013D 0C16    00628 Send1    movlw LOW1BIT         ; Send a '1' bit
013E 0976    00629          call  LineLow         ; hold the line low for 1/3rd of a Bit Cell
013F 0C32    00630          movlw HI_1BIT
0140 0983    00631          call  LineHi          ; let the line high for the rest of the Bit Cell
             00632
0141 0403    00633 Rotate   bcf    STATUS,C       ; Rotate out the MSB bit just sent from
0142 0360    00634          rlf    INDF,F         ; the Temporary Data Register
0143 02F7    00635          decfsz BitCntr,F      ; count down as bits are sent
0144 0B36    00636          goto   SndBits        ; loop until 8 bits are sent
0145 06B4    00637          btfsc  Flags1,F1Sent1 ; see whether all data has been sent
0146 0B4A    00638          goto   SndStop        ; if so, go send the Stop Bit
0147 05B4    00639          bsf    Flags1,F1Sent1 ; if not, set the Sent Flag,
0148 02A4    00640          incf   FSR,F          ; Then go prepare to send the next 8 bits,
0149 0B34    00641          goto   SetSend        ; and send the data from the next Data register
             00642
014A 0C38    00643 SndStop  movlw LOW0BIT         ; Send a '0' bit to the Host
014B 0976    00644          call  LineLow
014C 0C14    00645          movlw HI_0BIT
014D 0574    00646          bsf   Flags1,F1Stop   ; indicate to LineHi that this is the Stop
014E 0983    00647          call  LineHi         ; Bit let the line go high for 2/3rd of a Bit Cell
014F 04F4    00648          bcf   Flags1,F1Cllsn  ; a Collision did not occur, clear the flag
0150 0415    00649          bcf   Flags2,F2Srq    ; an Srq is no longer needed
0151 04D5    00650          bcf   Flags2,F2DSend  ; the Data has been sent
0152 0634    00651          btfsc Flags1,F1Reg3   ; If current Data Reg. is 3, don't allow
0153 0BC7    00652          goto  RunTsk2         ; Reg. 3 to be cleared (or at least the 1st 2
```


© 1997 Microchip Technology Inc.

```
0154 0213        00653          movf  RAMaddr,W      ; bytes) clear the Data Registers from which
0155 0024        00654          movwf FSR            ; the Data was sent via temporary registers
0156 0060        00655          clrf  INDF           ; Clear the registers holding the originalData
0157 02A4        00656          incf  FSR,F          ; which was just sent via the temporary regs.
0158 0060        00657          clrf  INDF           ; Go setup to run the 2nd Application Task for
0159 0BC7        00658          goto  RunTsk2        ; the time between the end of data sent, and
                 00659                               ; the beginning of the next Attention Signal
                 00660
015A 0634        00661 Collisn btfsc Flags1,F1Reg3  ; if there was a collision during a Talk
015B 05F4        00662          bsf   Flags1,F1C1lsn ; Reg. 3 Command, then set the Collision
015C 0AAB        00663          goto  AttnSig        ; Flag, otherwise, just abort to Attn Signal
                 00664
                 00665 ;*****************************************************************
                 00666
                 00667 ;*** RECEIVE DATA FROM THE HOST *** (Listen; calls Tlt, GetBit) ***
                 00668 ; Get the Tlt Signal (Stop to Start Time)
                 00669 ; Tlt recognizes the beginning of the Start Bit
                 00670 ; Load indirect address of temporary Data register
                 00671 ; Get the rest of the Start Bit
                 00672 ; Receive the first Data byte from the Host into the temporary Data
                 00673 ; register by calling GetBit - GetBit uses indirect address
                 00674 ; Set indirect address to 2nd temporary Data register
                 00675 ; Receive the second Data byte from the Host into the temporary Data
                 00676 ; register And then receive the Data Stop Bit if the
                 00677 ; data was not for Reg. 3, move the Data now stored in the temporary
                 00678 ; Data registers into the RAM locations of the Data register designated
                 00679 ; in RAMaddr, and go run the 2nd Application Task.
                 00680 ; if the data was for Reg. 3, go interpret what the Data Command was
                 00681 ; and take appropriate action.
                 00682
015D 0594        00683 Listen  bsf   Flags1,F1Lstn ; Set Listen Flag to tell Tlt (Stop to Start Time)
015E 094C        00684          call  Tlt          ; to look for the beginning of the Start Bit
015F 0C10        00685          movlw TmpReg1      ; receive bits into temporary registers
0160 0024        00686          movwf FSR          ; use indirect addressing to store received Data
0161 0060        00687          clrf  INDF         ; in temporary registers
0162 02A4        00688          incf  FSR,F        ;
0163 0060        00689          clrf  INDF         ; clear any data currently in temporary registers
0164 00E4        00690          decf  FSR,F        ;
0165 0C32        00691          movlw BIT_TST      ; load the TimeVariable to look for the rest of
0166 0039        00692          movwf TimeVar      ; the Start Bit
0167 0403        00693          bcf   STATUS,C     ; clear the Carry bit so it doesn't wrap around
0168 090C        00694          call  Get_Bit      ; get the rest of the Start bit
0169 0700        00695          btfss INDF,LSB     ; it should be a '1' bit
016A 0AAB        00696          goto  AttnSig      ; if not, abort to the Attn Signal
016B 0400        00697          bcf   INDF,LSB     ; don't let the Start Bit be the 1st bit of Data
016C 0C08        00698 SetRecv movlw BYTE          ; setup to receive 8 bits at a time into the reg.
016D 0037        00699          movwf BitCntr      ; count down as bits come in
016E 0C48        00700 RcvData movlw MAX_BIT       ; get & rotate a 1 or 0 bit into Data Reg., and
016F 0039        00701          movwf TimeVar      ; see if MAX_BIT time is exceeded & if so, abort
0170 0403        00702          bcf   STATUS,C     ; clear Carry bit so it doesn't wrap around
0171 0360        00703          rlf   INDF,F       ; rotate the bit into the Register (the 1st
0172 090C        00704          call  Get_Bit      ; rotation doesn't count)
0173 02F7        00705          decfsz BitCntr,F   ;decrement the counter each time a bit is
0174 0B6E        00706          goto  RcvData      ;received loop until 8 bits are received
0175 06D4        00707          btfsc Flags1,F1Rcvd1  ; see whether the 2nd Data byte was just
0176 0B7A        00708          goto  RcvStop      ;received if so, go get the Stop Bit
0177 05D4        00709          bsf   Flags1,F1Rcvd1  ; if not, set the Received-1st-Byte Flag,
0178 02A4        00710          incf  FSR,F        ;increment FSR to receive 2nd Byte of the Data
0179 0B6C        00711          goto  SetRecv      ;Reg. & go prepare to receive the next byte
                 00712
017A 0C48        00713 RcvStop movlw MAX_BIT       ;Get the '0' Stop Bit
017B 0039        00714          movwf TimeVar      ;
017C 0201        00715 RecvTmp movf  TMR0,W        ;Check the time, then check if the line went high
017D 0099        00716          subwf TimeVar,W    ;See if more than MAX_BIT usecs have passed
017E 0703        00717          btfss STATUS,C     ;if so, abort to Attn Signal
017F 0AAB        00718          goto  AttnSig      ;
```


© 1997 Microchip Technology Inc.

```
0180 0705    00719          btfss  PORTA,ADB      ; if not, check whether the line went high
0181 0B7C    00720          goto   RecvTmp        ; if still low, loop to check the time again
0182 0C32    00721          movlw  BIT_TST        ; if high, make sure the Stop Bit was '0'
0183 0039    00722          movwf  TimeVar        ; if the time was < BIT_TST, abort to
0184 0201    00723          movf   TMR0,W         ; the Attn Signal
0185 0099    00724          subwf  TimeVar,W      ; if the time was > BIT_TST, the '0' Stop
0186 0603    00725          btfsc  STATUS,C       ; Bit was received
0187 0AAB    00726          goto   AttnSig        ; clear TMR0 so second Task may use idle time
             00727
0188 0061    00728 RcvdDat  clrf   TMR0           ; Move Data to registers (unless for Reg 3.)
0189 0634    00729          btfsc  Flags1,F1Reg3  ; see if Data was received for Register 3,
018A 0B94    00730          goto   DataChk        ; if so, go interpret the Listen Reg. 3
018B 0213    00731          movf   RAMaddr,W      ; Command if not, move the received Data bytes
018C 0024    00732          movwf  FSR            ; to their indicated registers using indirect
018D 0210    00733          movf   TmpReg1,W      ; address,
018E 0020    00734          movwf  INDF
018F 02A4    00735          incf   FSR,F
0190 0211    00736          movf   TmpReg2,W
0191 0020    00737          movwf  INDF
0192 05B5    00738          bsf    Flags2,F2DRcvd; set the Data-has-been-received flag,
0193 0BC7    00739          goto   RunTsk2        ; and go prepare to run the 2nd Application Task
             00740
             00741 ;*************************************************************************
             00742
             00743 ;* INTERPRET THE LISTEN REG. 3 COMMAND SENT BY THE HOST *** (DataChk) *
             00744 ; This interprets the Data received for Register 3 as one of the
             00745 ; following Commands and runs the corresponding routine:
             00746 ;
             00747 ; Mask the Data Command received using the following Constants passed
             00748 ; to the IntData (Interpret Data Command) macro:
             00749 ; SELFTST (FF) - the Device is instructed to do a Self-Test
             00750 ; LISTEN1 (00) - unconditionally change Device Address and/or Status bits
             00751 ; LISTEN2 (FE) - change only the Device Address, and only change it
             00752 ;                 if the Device Address is marked as movable
             00753 ; DEV_ACT (FD) - change Device Address only if the Device Activator is
             00754 ;                 pressed (as defined in Device specification)
             00755
             00756 DataChk  IntData SELFTST,SlfTest ; see if Data Command is for Self Test
0194 0211        M          movf   TmpReg2,W       ; interprets the Data Command received by
0195 0FF         M          xorlw  SELFTST         ; comparing the 2nd byte to a Data
0196 0643        M          btfsc  STATUS,Z        ; Command constant
0197 0BA7        M          goto   SlfTest         ; it then goes to the appropriate routine
             00757          IntData LISTEN1,UpDat3a ; update bits Address and Status Bits (8
0198 0211        M          movf   TmpReg2,W       ; to 13) interprets the Data Command
0199 0F00        M          xorlw  LISTEN1         ; received by comparing the 2nd byte to a Data
019A 0643        M          btfsc  STATUS,Z        ; Command constant
019B 0BA9        M          goto   UpDat3a         ; it then goes to the appropriate routine
             00758          IntData LISTEN2,NewAddr ; change the Device Address (Bits 8 to 12)
019C 0211        M          movf   TmpReg2,W       ; interprets the Data Command received by
019D 0FFE        M          xorlw  LISTEN2         ; comparing the 2nd byte to a Data
019E 0643        M          btfsc  STATUS,Z        ; Command constant
019F 0BAF        M          goto   NewAddr         ; it then goes to the appropriate routine
             00759          IntData DEV_ACT,DevActv ; change the Device Address if the Device
01A0 0211        M          movf   TmpReg2,W       ; interprets the Data Command received by
01A1 0FFD        M          xorlw  DEV_ACT         ; comparing the 2nd byte to a Data
01A2 0643        M          btfsc  STATUS,Z        ; Command constant
01A3 0BAD        M          goto   DevActv         ; it then goes to the appropriate routine
             00760                                 ; Activator was pressed
01A4 0211    00761          movf   TmpReg2,W    ; if none of these Commands were given, put
01A5 002F    00762          movwf  Reg3b        ; received Data into Reg. 3b as a new Device the
01A6 0BC7    00763          goto   RunTsk2      ; Handler ID and go prepare to run the 2nd Task
             00764
01A7 0575    00765 SlfTest  bsf    Flags2,F2STest  ; Tell Device to do a Self-Test during 2nd
01A8 0BC7    00766          goto   RunTsk2         ; Task, and go prepare to run the 2nd Task
             00767
01A9 0210    00768 UpDat3a  movf   TmpReg1,W       ; Unconditionally change the Device Address
```


© 1997 Microchip Technology Inc.

```
                00769                           ; and/or the Status Bits of Reg. 3a
01AA 05C0       00770           bsf   W,ExpEvnt ; NOTE: Exceptional Event should remain as
01AB 002E       00771           movwf Reg3a     ; set to a '1' unless otherwise indicated
01AC 0BC7       00772           goto  RunTsk2   ; Go prepare to run the 2nd Application Task
                00773
01AD 0755       00774 DevActv btfss Flags2,F2DActv; if the Device Activator was NOT pressed,
01AE 0BC7       00775           goto  RunTsk2   ; go run the 2nd Application Task,
                00776                           ; if it was, change Device Address, if movable
01AF 06F4       00777 NewAddr btfsc Flags1,F1Cllsn; If a collison occurred during the last
01B0 0AAB       00778           goto  AttnSig   ; Talk Reg. 3, the Address was marked unmov
01B1 0210       00779           movf  TmpReg1,W ; able, abort to the Attention Signal.
01B2 0F00       00780           xorlw FALSE     ;
01B3 0643       00781           btfsc STATUS,Z
01B4 0AAB       00782           goto  AttnSig
01B5 020E       00783           movf  Reg3a,W   ; Create the new Device Address by masking in
01B6 0EF0       00784           andlw HI_NIBL   ; the Address received by the host, not allowing
01B7 0031       00785           movwf TmpReg2   ;  the upper nibble Status Bits in Reg. 3a to
01B8 0210       00786           movf  TmpReg1,W ;  be affected.
01B9 0E0F       00787           andlw LOW_NBL   ;
01BA 0111       00788           iorwf TmpReg2,W ; NOTE: Exceptional Event should remain as
01BB 05C0       00789           bsf   W,ExpEvnt ; set to a '1' unless otherwise indicated
01BC 002E       00790           movwf Reg3a     ;   when the new Device Address is in place,
01BD 0BC7       00791           goto  RunTsk2   ; go prepare to run the 2nd Application Task
                00792
                00793 ;*************************************************************************
                00794 ;*** FLUSH THE REGISTER SPECIFIED BY THE COMMAND BYTE *** (Flush) ***
                00795
01BE 0213       00796 Flush   movf    RAMaddr,W ; Clear the Data in the specified Register
01BF 0024       00797         movwf   FSR       ; use indirect address to clear the RAM
01C0 0060       00798         clrf    INDF      ; locations holding the Data
01C1 02A4       00799         incf    FSR,F
01C2 0060       00800         clrf    INDF
01C3 0BC7       00801         goto    RunTsk2
                00802
                00803 ;*************************************************************************
                00804
01C4 0AAB       00805 Reserv1 goto    AttnSig   ; No action until Reserved Command 1 is defined
                00806
                00807 ;*************************************************************************
                00808
01C5 0AAB       00809 Reserv2 goto    AttnSig   ; No action until Reserved Command 2 is defined
                00810
                00811 ;*************************************************************************
                00812
01C6 0AAB       00813 Reserv3 goto    AttnSig   ; No action until Reserved Command 3 is defined
                00814
                00815 ;*************************************************************************
                00816
                00817 ;*** PUT THE CODE FOR OTHER APPLICATION HERE *** (RunTsk2, Task_2) ***
                00818
                00819 ;         bsf   Flags2,F2SFail ; code would go before here if a Self Test
                00820 ;         bcf   Flags2,F2SFail ; was performed and it failed or passed
                00821
01C7 0070       00822 RunTsk2 clrf  TmpReg1       ; When finished with Data interpretation,
01C8 0071       00823         clrf  TmpReg2       ; clear the temporary Data registers, and
01C9 0CE1       00824         movlw TSK2MAX       ; load Task 2 TimeVariable with amount allowed
01CA 003A       00825         movwf Tsk2Var       ; between end of Data and Attention Signal. If
                00826
01CB 0615       00827 Task_2  btfsc Flags2,F2Srq  ; the Srq Flag has not been cleared, then data
01CC 0BF7       00828         goto  AttnTst       ; must still be sent from 1st Service Request
01CD 0900       00829         call  PrScale       ; Turn on the TMR0 prescale for >250usec count
                00830
01CE 0675       00831 Tests   btfsc Flags2,F2STest ; See if Key-Up transition codes should be
01CF 0BDC       00832         goto  LoadDat       ; sent
01D0 0635       00833         btfsc Flags2,Switch ; Determine if the Switch has been
01D1 0BED       00834         goto  DBounce       ; de-bounced if not, go timeout
```

```
01D2 0626      00835          btfsc   PORTB,Switch  ; Check if Switch is pressed,
01D3 0BF3      00836          goto    Tsk2Tmp       ; if not, go timeout
01D4 05F5      00837          bsf     Flags2,F2DMore; data needs to be sent to the host
01D5 0515      00838          bsf     Flags2,F2Srq  ; and issue a Service Request
01D6 0535      00839          bsf     Flags2,Switch ; set the flag for de-bouncing switch
01D7 0506      00840          bsf     PORTB,LED     ; Turn on LED when Switch is pressed
01D8 0C08      00841          movlw   DEBOUNC
01D9 003B      00842          movwf   TmpCtr1
01DA 06D5      00843          btfsc   Flags2,F2DSend; The last Data was sent correctly if Talk
01DB 0BF3      00844          goto    Tsk2Tmp       ; cleared the DSend flag, if set, goto
               00845                                ; Attn Test to re-send Data
               00846
01DC 07F5      00847 LoadDat  btfss   Flags2,F2DMore; If all the Data has been sent, DMore is
01DD 0BF3      00848          goto    Tsk2Tmp       ; clearif DMore is clear, go time out
01DE 0C38      00849          movlw   SHIFT         ; if DMore is set, Data remains to be sent
01DF 0028      00850          movwf   Reg0a         ; if not, load the Data bytes
01E0 0C12      00851          movlw   BANG
01E1 0029      00852          movwf   Reg0b
01E2 05D5      00853          bsf     Flags2,F2DSend; Data now needs to be sent to the host
01E3 0515      00854          bsf     Flags2,F2Srq  ; Until all data has been sent, Srq's may
01E4 0675      00855          btfsc   Flags2,F2STest; be sent See if Key-Up Transition Codes
01E5 0BE8      00856          goto    KeyUp         ; should be sent if so, go set the bits
01E6 0575      00857          bsf     Flags2,F2STest; if not, set bit so they will be next time
01E7 0BED      00858          goto    DBounce       ;  and go debounce the switch
               00859
01E8 05E8      00860 KeyUp    bsf     Reg0a,07h     ; Set the 7th bit in each register to
01E9 05E9      00861          bsf     Reg0b,07h     ; indicate the Key is up
01EA 0475      00862          bcf     Flags2,F2STest; The Key-Up Transition Code bits have been
01EB 04F5      00863          bcf     Flags2,F2DMore; set All data will have been sent to the
01EC 0BED      00864          goto    DBounce       ; host after this transaction
               00865
01ED 0726      00866 DBounce  btfss   PORTB,Switch  ; Check if Switch has been released,
01EE 0BF3      00867          goto    Tsk2Tmp       ; if not, go timeout
01EF 02FB      00868          decfsz  TmpCtr1,F     ; if so, start timed debounce of several
01F0 0BF3      00869          goto    Tsk2Tmp       ; millisecs. before switch is tested again
01F1 0406      00870          bcf     PORTB,LED     ; Turn off LED when Switch is released
01F2 0435      00871          bcf     Flags2,Switch ; clear de-bounce flag
               00872
01F3 0201      00873 Tsk2Tmp  movf    TMR0,W      ; Check the time to see if more than the maximum
01F4 009A      00874          subwf   Tsk2Var,W   ; time limit has been exceeded
01F5 0603      00875          btfsc   STATUS,C    ; if so, go determine what part of Attn Signal
01F6 0BF3      00876          goto    Tsk2Tmp
               00877
01F7 0714      00878 AttnTst  btfss   Flags1,F1Attn ; After this portion of the 2nd Task is
01F8 0AAB      00879          goto    AttnSig       ; complete,If 2nd Task is NOT run during
01F9 0414      00880          bcf     Flags1,F1Attn ; Attn Signal, go get the start of the Attn
01FA 0AC2      00881          goto    AttnMin       ; Signal otherwise, go get the rest of the
               00882                                ; Attn Signal
               00883 ;*************************************************************************
               00884
01FF           00885          ORG     PIC54
01FF 0A90      00886 RESETV   goto    Start
               00887
               00888      END
```


© 1997 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXX----X

All other memory blocks unused.

Program Memory Words Used:   508
Program Memory Words Free:   516


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN594

Using the CCP Module(s)

Author: *Mark Palmer*
Microchip Technology Inc.

This application note discusses the operation of a Capture/Compare/PWM (CCP) module, and the interaction of multiple CCP modules with the timer resources.

The (CCP) module is software programmable to operate in one of three modes:

1. A Capture input
2. A Compare output
3. A Pulse Width Modulation (PWM) output

For the CCP module to function, Timer resources must be used in conjunction with the CCP module. The desired CCP mode of operation determines which timer resources are required. Table 1 shows the CCP mode with the corresponding timer resource required. Both the Capture and Compare modes require that Timer1 be operating in timer mode or synchronized counter mode.

Note: Capture and Compare modes may not operate if Timer1 is operated in the asynchronous counter mode.

TABLE 1: CCP MODE - TIMER RESOURCE

CCP MODE	Timer Resource
Capture	Timer1
Compare	Timer1
PWM	Timer2

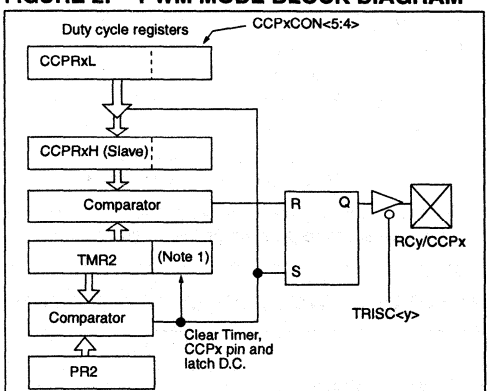
CCP OPERATION

The following three sections discuss the operation of the CCP module in each of its modes of operation. There is a simple example program for each mode of operation. The software example for the Capture mode, also uses a second CCP module in Compare mode to generate the signal to capture.

PWM Mode

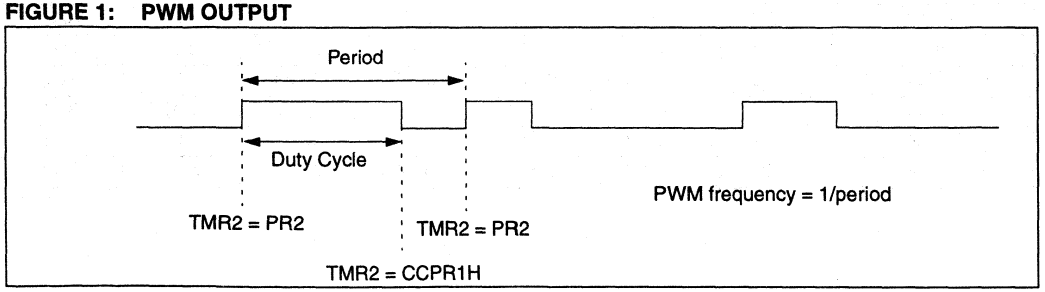
A Pulse Width Modulation output (Figure 1) is a signal that has a time-base (period) and a time that the output stays high (duty cycle). The period is the duration after which the PWM rising edge repeats itself. The resolution of the PWM output is the granularity with which the duty cycle can be varied. The frequency of a PWM is simply the inverse of the period (1/ period).

FIGURE 2: PWM MODE BLOCK DIAGRAM



Note: 8-bit timer is concatenated with 2-bit internal Q clock or 2 bits of the prescaler to create 10-bit time-base.

FIGURE 1: PWM OUTPUT



© 1997 Microchip Technology Inc.

Each CCP module can support one Pulse Width Modulation (PWM) output signal, with minimal software overhead. This PWM signal can attain a resolution of up to 10-bits, from the 8-bit Timer2 module. This gives 1024 steps of variance from an 8-bit overflow counter. This gives a maximum accuracy of Tosc (50 ns, when the device is operated at 20 MHz). Figure 2 shows a block diagram of the CCP module in PWM mode. When the Timer2 overflows (timer = period register), the value in the duty cycle registers (CCPRxL:CCPRxCON<5:4>) is latched into the 10-bit slave latch. A new duty cycle value can be loaded into the duty cycle register(s) at any time, but is only latched into the slave latch when Timer2 = Timer2 Period Register (PR2).

The period of Timer 2 (and PWM) is determined by the frequency of the device, the Timer2 prescaler value (1, 4 or 16), and the Timer2 Period Register. Equation 1 shows the calculation of the PWM period, duty cycle, and the minimum and maximum frequencies.

Appendix A is a program which generates up to a 10-bit PWM output. The PWM period and duty cycle are updated after the overflow of Timer1. Upon the overflow of Timer1, ports A, B and D are read. The 10-bit duty cycle is specified by the value on PORTB:PORTA<1:0>, while the period is specified by the value on PORTD. By setting the conditional assemble flag `PICMaster` to `TRUE`, these values are read from internal registers which are dummy registers for the ports (DUMMY_Px). This allows the software to be verified without the use of hardware and external stimulus.

Since the PWM duty cycle is double buffered, the duty cycle registers are only loaded when there is sufficient time to complete the update of the 10-bit value before the Timer2 = PR2 match occurs. After the duty cycle has been updated and the Timer2 = PR2 match has occurred, the period (stored in the PR2 register) is updated. The operation of the CCP module in PWM mode is similar to the PIC17C42's PWM. Additional concepts of PWM operation can be found in Application Notes AN564 and AN539.

EQUATION 1: PWM PERIOD, DUTY CYCLE, AND FREQUENCIES

PWM Period = [(PR2) + 1] • 4 TOSC • (Timer 2 prescale value)

PWM Duty Cycle = [CCPRxL:CCPRxCON<5:4>] • 4 TOSC • (Timer2 prescale value)

PWM maximum frequency
(High Resolution mode) = 4/(PR2 • TCY)
(Low Resolution mode) = 1/(PR2 • TCY)

PWM minimum frequency
(High Resolution mode) = 4/(PR2 • TCY)
(Low Resolution mode) = 1/(PR2 • TCY)

Table 2 shows the minimum and maximum PWM frequency for different device frequencies. The Timer2 prescaler will be selected to give either the minimum or maximum frequencies as shown.

TABLE 2: PWM FREQUENCY FOR DIFFERENT DEVICE FREQUENCIES

PWM Resolution	20 MHz		10 MHz		2 MHz		Units
	Min	Max	Min	Max	Min	Max	
10-bit	1.22	19.53	0.613	9.77	0.123	1.96	kHz
9-bit	1.22	39.06	0.613	9.77	0.123	3.92	kHz
10-bit	1.22	19.53	0.613	9.77	0.123	1.96	kHz
9-bit	1.22	39.06	0.613	9.77	0.123	3.92	kHz
10-bit	1.22	19.53	0.613	9.77	0.123	1.96	kHz
9-bit	1.22	39.06	0.613	9.77	0.123	3.92	kHz
10-bit	1.22	19.53	0.613	9.77	0.123	1.96	kHz
9-bit	1.22	39.06	0.613	9.77	0.123	3.92	kHz

© 1997 Microchip Technology Inc.

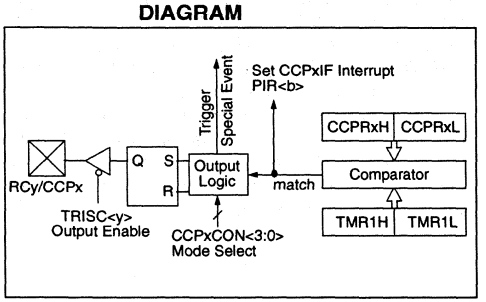
Compare Mode

In Compare mode, the 16-bit value of Timer1 is compared to the CCPRxH:CCPRxL registers. When these registers match, the S/W configured event occurs on the CCPx pin. The events that can be S/W selected are:

- Clear CCPx pin on match
- Set CCPx pin on match
- Generate S/W interrupt (CCPx pin unchanged)
- Trigger special event (CCPx pin unchanged)
  - CCP1 clears Timer1
  - CCP2 clears Timer1 and sets the A/D's GO/$\overline{\text{DONE}}$ bit

The CCPxM3:CCPxM0 control bits, in register CCPxCON, configures the operation of the CCP module. The compare function must have the data direction of the CCPx pin configured as an output, if the compare event is to control the state of the CCPx pin.

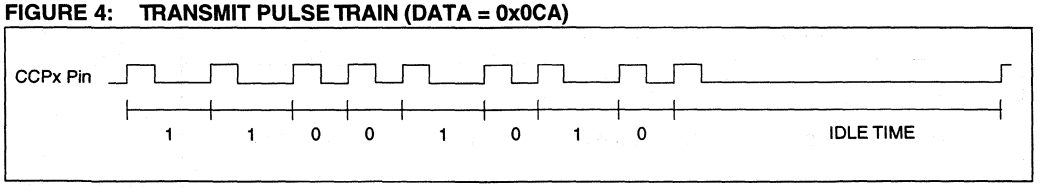
FIGURE 3: COMPARE MODE BLOCK DIAGRAM



When the CCP module is in the OFF state (CCPxM3:CCPxM0 = 0h), the CCPx output latch is forced to a low level, though the level on the CCPx pin will be determined by the value in the data latch of the port. Figure 3 shows the block diagram of the CCP module in Compare mode.

Appendix B is a program which uses the CCP module to transmit a pulse train dependent on the data byte. Timer1 is used as a free running timer, with each "new" compare value being an offset added to the present CCP compare latch value. The data is transmitted every 600 µs. Each data bit has a sync pulse (High level) of 8.8 µs. Then the data is transmitted as a low pulse. The time duration of the low pulse determines the value of the data bit. A '0' bit is low for 18.8 µs while a '1' bit is low for 37.6 µs. After the last data bit has been transmitted, another sync pulse is transmitted and the output remains low (idle time) until the 600 µs data period has completed. An example of the pulse train for the a data byte of 0x0CA is shown in Figure 4, and has an idle time of 224 µs. These pulse times are based off the device operational frequency. The program header file, `COMP.H`, calculates the values to be loaded into the compare registers from the specified Device_freq. The data to be transmitted is read from PORTB, during the idle time. By setting the conditional assemble flag `PICMaster` to `TRUE`, these values are read from internal registers which are dummy registers for the ports (DUMMY_Px). This allows the software to be verified without the use of hardware and external stimulus.

FIGURE 4: TRANSMIT PULSE TRAIN (DATA = 0x0CA)



© 1997 Microchip Technology Inc.

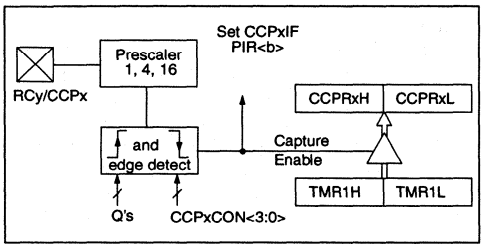
Capture Mode

In Capture mode, the 16-bit value of Timer1 is latched into the CCPRxH:CCPRxL registers, when the S/W configured event occurs on the CCPx pin. The events that can cause a capture are:

- Every falling edge
- Every rising edge
- Every 4th rising edge
- Every 16th rising edge

The CCPxM3:CCPxM0 control bits, in register CCPxCON, configures the operation of the CCP module. The capture function works regardless of the data direction of the CCPx pin (input or output). With the CCPx pin is configured as an output, a write to the CCPx pin (in PORTC) will cause a capture when the capture requirement is met.

FIGURE 5: CAPTURE MODE BLOCK DIAGRAM

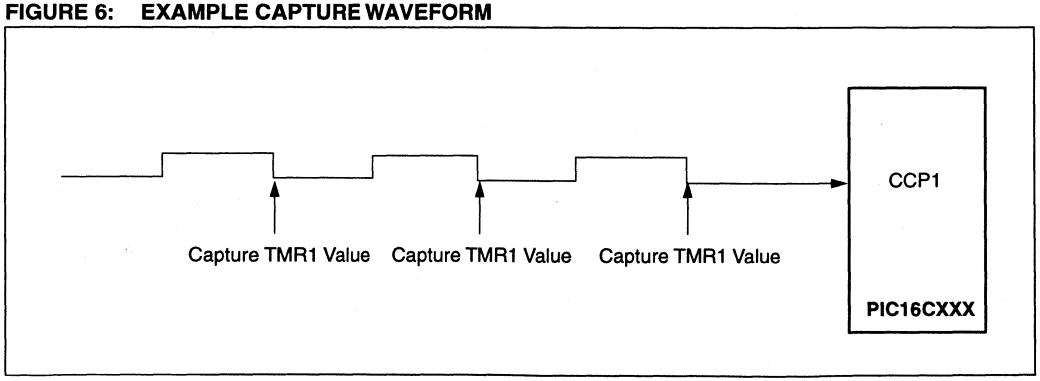


The changing of the Capture mode, via the CCPxM3:CCPxM0 bits, may cause the CCPxIF bit to be set. This "false" interrupt should be cleared (ignored) after changing between capture modes. The CCP prescaler is only cleared by configuring the CCP module into the OFF state (CCPxM3:CCPxM0 = 0h). Figure 5 shows the block diagram of the CCP module in Capture mode. The use of the CCP module in Capture mode is similar to the PIC17C42's capture. Additional concepts of capture operation can be found in Application Note AN545.

Appendix C is a program which implements a 16-bit capture from a free running timer (Timer1). The capture event is configured as each rising edge. The 16-bit capture value is the "new" 16-bit capture value minus the "old" 16-bit capture value. If the time between captures is greater than 2^{16} Timer1 increments, an invalid result will occur. This invalid result is not indicated by the software. After the capture period result is calculated, the "new" capture value is loaded into the "old" register.

The waveform that is captured is generated from a second CCP module in compare mode. The value that is loaded in to the CCPR2H:CCPR2L is read from the PORTB and PORTD registers. By setting the conditional assemble flag `PICMaster` to `TRUE`, these values are read from internal registers which are dummy registers for the ports (DUMMY_Px). This allows the software to be verified without the use of hardware and external stimulus. Figure 6 shows an input into the CCPx pin, and the capture measurement points.

FIGURE 6: EXAMPLE CAPTURE WAVEFORM



© 1997 Microchip Technology Inc.

INTERACTION OF CCP MODULES

Due to the modularity of the PIC16CXXX peripherals, future devices with two or more CCP modules on a device are possible. Each CCP module operates independently from the others, though their interaction with the timer resources must be taken into account.

When two or more CCP modules exist on a device, there can be an interaction between the CCP modules. This interaction is shown in Table 3. These interactions do NOT include any interaction (S/W) caused by the main program nor the interrupt service routines of the CCP sources.

Interaction of Two Capture Modes

When two CCP modules are in a Capture mode, Timer1 is the time-base for both captures. This means that they will have the same capture resolution, as determined by the Timer1 prescaler and frequency of the timer/counter clock. This clock can come from an external source (on the RC0/T1OSO/T1CKI pin), but must be synchronized to the device.

Interaction of One Capture Mode and One Compare Mode

When one CCP module is in a Capture mode and a second CCP module is in Compare mode, Timer1 is the time-base for both the captures and the compare. This means that the capture and the compare will have the same resolution, as determined by the Timer1 prescaler and frequency of the timer/counter clock. This clock can come from an external source (on the RC0/T1OSO/T1CKI pin), but must be synchronized to the processor clock. Also, care must be taken in that the compare can be configured to clear TMR1 register (when in special Trigger mode). Care must be taken in system design to ensure that this clearing of the TMR1 register does not have any negative impact on the capture function.

3

TABLE 3: INTERACTION OF TWO CCP MODULES

CCPx Mode	CCPy Mode	Interaction
Capture	Capture	Same Timer1 time-base.
Capture	Capture	The compare could be configured for trigger special event, which clears the TMR1 register.
Capture	Capture	The compare(s) could be configured for trigger special event, which clears the TMR1 register.
PWM	PWM	The PWMs will have the same frequency, and update rate (Timer2 interrupt).
PWM	Capture	None
PWM	Capture	None

© 1997 Microchip Technology Inc.

EXAMPLE 1:

ACTION	TIMER1 STATE	COMMENT
CCPR1H:CCPR1L = 0x0465	0x????	
CCP1CON = 0x?B	0x????	CCP1 in Compare - Special Trigger Mode
:		
:	0x0232	
:		
CCPR2H:CCPR2L = 0x0165	0x0333	
CCP2CON = 0x?B	0x0334	CCP2 in Compare - Special Trigger Mode
:		
	0x0465	CCP1 resets TMR1 and CCP1 -
	0x0000	Special Trigger function occurs
:		
	0x0165	CCP2 resets TMR1 and CCP2 -
	0x0000	Special Trigger function occurs
:		
	0x0165	CCP2 resets TMR1 and CCP2 -
	0x0000	Special Trigger function occurs
:		

Interaction of Two Compare Modes

When two CCP modules are in a Compare mode, Timer1 is the time-base for both compares. This means that they will have the same compare resolution, as determined by the Timer1 prescaler and frequency of the timer/counter clock. This clock can come from an external source (on the RC0/T1OSO/T1CKI pin), but must be synchronized to the processor clock. Since the compare modules can be configured to clear TMR1 register (when in special Trigger mode), care must be taken in system design to ensure that this clearing of the TMR1 register does not have any negative impact on the compare function. If both compares are configured with a special trigger, which clears the TMR1 register, then the compare register that is closest to (but greater than) the TMR1 register value is the compare value that will reset the TMR1 register. Example 1 shows a possible case.

Interaction of Two PWM Modes

When two CCP modules are in a PWM mode, Timer2 is the time-base for both PWM outputs. This means that they will have the same PWM frequency and update rates, as determined by the Timer2 prescaler and frequency of the device. The resolution of the two PWMs may be different, since each CCP module has its own CCPxX:CCPxY bits for high resolution mode. These bits are found in the CCPxCON<5:4> register.

CONCLUSION

The Capture/Compare/PWM modules offer enormous flexibility in the use of the device timer resources. As with all resources, care must be taken to ensure that no adverse system complications can occur with the interaction between multiple CCP modules. The programs for simple operation of the various CCP modes should be a good foundation for modifications to suite your particular needs.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40 Released            PWM_1.ASM   1-16-1997  17:35:52         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C74, n = 66
                     00002         ERRORLEVEL  -302
                     00003 ;
                     00004 ;*****************************************************************
                     00005 ;
                     00006 ; This program outputs a PWM signal on the CCP1 pin. The duty cycle and
                     00007 ; period of the PWM is read every time TMR1 overflows.
                     00008 ;    PERIOD     = PORTB
                     00009 ;    DUTY CYCLE = PORTD and PORTE<1:0>
                     00010 ;
                     00011 ; The prescaler of TMR2 is selected by the state of PORTA<1:0> after
                     00012 ; reset    RA1:RA0    Prescaler multiplies Tcyc by
                     00013 ;            0 0       1
                     00014 ;            0 1       4
                     00015 ;            1 x      16
                     00016 ;
                     00017 ;
                     00018 ;       Program = PWM_1.ASM
                     00019 ;       Revision Date:   7-13-94
                     00020 ;                        1-16-97      Compatibility with MPASMWIN 1.40
                     00021 ;
                     00022 ;
                     00023 ;*****************************************************************
                     00024 ;
                     00025 ;
                     00026 ; HARDWARE SETUP
                     00027 ;    PORTA<1:0>   -  Prescaler to TMR2, read only after reset
                     00028 ;    PORTB        -  Period of PWM
                     00029 ;    PORTD        -  Duty Cycle high of PWM (8-bits)
                     00030 ;    PORTE<1:0>   -  Duty Cycle low of PWM (2-bits)
                     00031 ;
                     00032 ;
                     00033             INCLUDE <p16c74.inc>
                     00001         LIST
                     00002 ; P16C74.INC Standard Header File, Ver. 1.00 Microchip Technology, Inc.
                     00318         LIST
                     00034
  00000000           00035 FALSE           EQU     0
  00000001           00036 TRUE            EQU     1
                     00037
                     00038             INCLUDE <PWM.h>
                     00046         list
                     00047
                     00039 ;
  00000001           00040 PICMaster       EQU     TRUE      ; A Debugging Flag
  00000001           00041 Debug           EQU     TRUE      ; A Debugging Flag
  00000001           00042 Debug_PU        EQU     TRUE      ; A Debugging Flag
                     00043 ;
                     00044 ;
                     00045 ; Reset address. Determine type of RESET
                     00046 ;
0000                 00047       org     RESET_V           ; RESET vector location
0000 1683            00048 RESET     BSF     STATUS, RP0   ; Bank 1
0001 188E            00049           BTFSC   PCON,NOT_POR  ; Power-up reset?
```

```
0002 2832          00050           GOTO    START           ; YES
0003 285C          00051           GOTO    OTHER_RESET     ; NO, a WDT or MCLR reset
                   00052 ;
                   00053 ; This is the Periperal Interrupt routine. Need to determine the type
                   00054 ; of interrupt that occurred. The following interrupts are enabled:
                   00055 ;   1.  CCP Capture Occured
                   00056 ;
                   00057     page
0004               00058       org     ISR_V               ; Interrupt vector location
0004               00059 PER_INT_V
0004 1283          00060           BCF     STATUS, RP0     ; Bank 0
0005 180C          00061           BTFSC   PIR1, TMR1IF    ; TMR1 Overflow Interrupt occured?
0006 280D          00062           GOTO    T1OVFL          ; YES, Service the TMR1 Interrupt
0007               00063 ERROR1                            ; NO, Error Condition-Unknown Interrupt
0007 1505          00064           BSF     PORTA, 2        ; Toggle a PORT pin
0008 1105          00065           BCF     PORTA, 2
0009 2807          00066           GOTO    ERROR1
                   00067 ;
000A               00068 ERROR2                            ; NO, Error Condition-Unknown Interrupt
000A 1585          00069           BSF     PORTA, 3        ; Toggle a PORT pin
000B 1185          00070           BCF     PORTA, 3
000C 280A          00071           GOTO    ERROR2
                   00072 ;
000D               00073 T1OVFL
000D 100C          00074           BCF     PIR1, TMR1IF    ; Clear T1 Overflow Interrupt Flag
                   00075       if (PICMaster )
000E 0853          00076           MOVF    DUMMY_PD, W     ;
                   00077       else
                   00078           MOVF    PORTD, W        ;
                   00079       endif
000F 00D5          00080           MOVWF   DC_HI           ;
                   00081       if (PICMaster )
0010 0854          00082           MOVF    DUMMY_PE, W     ;
                   00083       else
                   00084           MOVF    PORTE, W        ;
                   00085       endif
0011 00D6          00086           MOVWF   DC_LO           ;
                   00087       if (PICMaster )
0012 0851          00088           MOVF    DUMMY_PB, W     ;
                   00089       else
                   00090           MOVF    PORTB, W        ;
                   00091       endif
0013 1683          00092           BSF     STATUS, RP0     ; Bank 1
0014 00A0          00093           MOVWF   T2_PERIOD       ;
0015 1283          00094           BCF     STATUS, RP0     ; Bank 0
                   00095 ;
0016               00096 WAIT_DC
0016 0811          00097           MOVF    TMR2, W         ; Read present TMR2 register value
0017 0212          00098           SUBWF   PR2, W          ; How close is the timer to rolling over
0018 390F          00099           ANDLW   0x0F            ; Does this make it zero?
0019 1903          00100           BTFSC   STATUS, Z       ; If Z is set, near rollover
001A 2816          00101           GOTO    WAIT_DC         ; loop until rolled over
001B 0855          00102           MOVF    DC_HI, W        ; else losd the duty cycle values
001C 0095          00103           MOVWF   CCPR1L          ; Load DC high
001D 300F          00104           MOVLW   0x0F            ;
001E 0597          00105           ANDWF   CCP1CON, F      ; Set the DC low bits
001F 18D6          00106           BTFSC   DC_LO, 1        ;
0020 1697          00107           BSF     CCP1CON, CCP1X;
0021 1856          00108           BTFSC   DC_LO, 0        ;
0022 1617          00109           BSF     CCP1CON, CCP1Y;
0023 108C          00110           BCF     PIR1, TMR2IF    ; Clear the TRM2 = PR2 flag
                   00111 ;
0024               00112 WAIT_PR
0024 1C8C          00113           BTFSS   PIR1, TMR2IF    ; LOOP waiting for TRM2 = PR2
0025 2824          00114           GOTO    WAIT_PR         ; Need to wait until TMR2 = PR2 so that
                   00115                                   ;  Duty Cycle is latched.
```


© 1997 Microchip Technology Inc.

```
0026 1683           00116           BSF     STATUS, RP0  ; Bank 1
0027 300F           00117           MOVLW   0x0F         ; Load TMR2 period with minimum value Fh
0028 0092           00118           MOVWF   PR2          ;
0029 30F0           00119           MOVLW   0xF0         ;
002A 0520           00120           ANDWF   T2_PERIOD, W ; Determine if period needs to be greater
002B 1903           00121           BTFSC   STATUS, Z    ;
002C 2830           00122           GOTO    NO_OFFSET    ; NO, Period is the minimum
002D                00123 PR_OFFSET
002D 300F           00124           MOVLW   0x0F         ; Yes, calculate additional offset
002E 0220           00125           SUBWF   T2_PERIOD, W ;
002F 0792           00126           ADDWF   PR2, F       ; ADD Period offset
                    00127 ;
0030                00128 NO_OFFSET
0030 1283           00129           BCF     STATUS, RP0  ; Bank 0
0031 0009           00130           RETFIE               ; Return / Enable Global Interrupts
                    00131 ;
                    00132     page
                    00133 ;
                    00134 ;*************************************************************************
                    00135 ;*****      Start program here, Power-On Reset occurred.
                    00136 ;*************************************************************************
                    00137 ;
0032                00138 START                          ; POWER_ON Reset (Beginning of program)
0032 1283           00139           BCF     STATUS, RP0  ; Bank 0
0033 018F           00140           CLRF    TMR1H        ;
0034 018E           00141           CLRF    TMR1L        ;
                    00142 ;
0035                00143 MCLR_RESET                     ; A Master Clear Reset
0035 0183           00144           CLRF    STATUS       ; Do initialization (Bank 0)
0036 018B           00145           CLRF    INTCON
0037 018C           00146           CLRF    PIR1
0038 1683           00147           BSF     STATUS, RP0  ; Bank 1
0039 3080           00148           MOVLW   0x80         ;
003A 0081           00149           MOVWF   OPTION_REG   ;
003B 018C           00150           CLRF    PIE1         ; Disable all peripheral interrupts
003C 30FF           00151           MOVLW   0xFF         ;
003D 009F           00152           MOVWF   ADCON1       ; Port A is Digital.
                    00153 ;
                    00154 ;
003E 1283           00155           BCF     STATUS, RP0  ; Bank 0
003F 0185           00156           CLRF    PORTA        ; ALL PORT output should output Low.
0040 0186           00157           CLRF    PORTB
0041 0187           00158           CLRF    PORTC
0042 0188           00159           CLRF    PORTD
0043 0189           00160           CLRF    PORTE
                    00161 ;
0044 1683           00162           BSF     STATUS, RP0  ; Select Bank 1
0045 30FF           00163           MOVLW   0xFF         ;
0046 0085           00164           MOVWF   TRISA        ; RA5 -  0 inputs
0047 0086           00165           MOVWF   TRISB        ; RB7 - 0 inputs
0048 0187           00166           CLRF    TRISC        ; RC Port are outputs
0049 0088           00167           MOVWF   TRISD        ; RD Port are inputs
004A 0089           00168           MOVWF   TRISE        ; RE Port are inputs
004B 0092           00169           MOVWF   PR2          ; Default PWM period
004C 140C           00170           BSF     PIE1, TMR1IE ; Enable TMR1 Interrupt
004D 1283           00171           BCF     STATUS, RP0  ; Select Bank 0
                    00172 ;
004E 300C           00173           MOVLW   0X0C         ; CCP module is in
004F 0097           00174           MOVWF   CCP1CON      ; PWM output mode
                    00175 ;
                    00176 ; Initialize the Special Function Registers (SFR) interrupts
                    00177 ;
0050 018C           00178           CLRF    PIR1         ;
0051 0190           00179           CLRF    T1CON        ;
0052 0192           00180           CLRF    T2CON        ;
                    00181       if (PICMaster )
```

```
0053 1850           00182           BTFSC   DUMMY_PA, 0   ;
                    00183       else
                    00184           BTFSC   PORTA, 0      ;
                    00185       endif
0054 1412           00186           BSF     T2CON, 0      ;
                    00187 ;
                    00188       if (PICMaster )
0055 18D0           00189           BTFSC   DUMMY_PA, 1   ;
                    00190       else
                    00191           BTFSC   PORTA, 1      ;
                    00192       endif
0056 1492           00193           BSF     T2CON, 1      ;
                    00194 ;
0057 170B           00195           BSF     INTCON, PEIE  ; Enable Peripheral Interrupts
0058 178B           00196           BSF     INTCON, GIE   ; Enable all Interrupts
0059 1410           00197           BSF     T1CON, TMR1ON ; Turn Timer 1 ON
005A 1512           00198           BSF     T2CON, TMR2ON ; Turn Timer 2 ON
                    00199 ;
005B 285B           00200 lzz       goto    lzz           ; Loop waiting for TMR1 interrupt
                    00201 ;
                    00202 ; Here is where you do things depending on the type of RESET (Not a
                    00203 ; Power-On Reset).
005C 1E03           00204 OTHER_RESET BTFSS STATUS,NOT_TO ; WDT Time-out?
005D 2807           00205 WDT_TIMEOUT GOTO   ERROR1       ; YES, This is error condition
                    00206       if ( Debug_PU )
005E 2832           00207           goto   START          ; MCLR reset, Goto START
                    00208       else
                    00209           GOTO   MCLR_RESET     ; MCLR reset, Goto MCLR_RESET
                    00210       endif
                    00211 ;
                    00212       if (Debug )
005F 0000           00213 END_START   NOP                 ; END label for debug
                    00214       endif
                    00215 ;
                    00216 ;
07FF                00217     org     PMEM_END          ; End of Program Memory
07FF 2807           00218             GOTO    ERROR1    ; If you get here your program was lost
                    00219
                    00220     end
_XT_OSC                         00003FBD
__16C74                         00000001
lzz                             0000005B


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX ---------------- ----------------
07C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:    97
Program Memory Words Free:  3999


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,    17 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: PWM.H

```
    nolist
;*************************************************************************
;
;  This is the custom Header File for the real time clock application note
;   PROGRAM:CLOCK.H
;   Revision:7-13-94
;
;*************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq        EQU     D'10000000'; Device Frequency is 4 MHz
PULSE_TIME      EQU     (( Dev_Freq / D'4000' ) * D'188' / D'10000' )
;

DB_HI_BYTE      EQU     (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY  EQU     (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR      EQU     40      ; RAM Location
OUTER_CNTR      EQU     41      ; RAM Location
;
T1OSO           EQU     0       ; The RC0 / T1OSO / T1CKI
;
RESET_V         EQU     0x0000 ; Address of RESET Vector
ISR_V           EQU     0x0004 ; Address of Interrupt Vector
PMEM_END        EQU     0x07FF ; Last address in Program Memory
TABLE_ADDR      EQU     0x0400 ; Address where to start Tables
;
COUNTER         EQU     0x021       ;
;
XMIT_DATA       EQU     0x30
DATA_CNT        EQU     0x31
ONES_CNT        EQU     0x32
CCP1_INT_CNT    EQU     0x33
CCPREG_HI       EQU     0x40
CCPREG_LO       EQU     0x41
DUMMY_PA        EQU     0x50
DUMMY_PB        EQU     0x51
DUMMY_PC        EQU     0x52
DUMMY_PD        EQU     0x53
DUMMY_PE        EQU     0x54
DC_HI           EQU     0x55
DC_LO           EQU     0x56
T2_PERIOD       EQU     0xA0
;
    list
```

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C: COMP_1.LST

```
MPASM 01.40 Released          COMP_1.ASM   1-16-1997  17:35:21         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C74, n = 66
                     00002         ERRORLEVEL  -302
                     00003 ;
                     00004 ;*****************************************************************
                     00005 ;
                     00006 ; This program outputs a pulse train on the CCP1 pin, as specified by
                     00007 ; the values in the CCPR1H:CCPR1L.
                     00008 ;
                     00009 ;                                ____          ____      __
                     00010 ;  Pulse Train     ___|    |_____|    |     |____|
                     00011 ;  Data Value           <---- 1 ----> <-- 0 -->
                     00012 ;                         ^       ^       ^     ^
                     00013 ;                         |       |       |     |
                     00014 ;                         |  T_ONE_BIT    |  T_ZERO_BIT
                     00015 ;                         |               |
                     00016 ;                    PULSE_TIME      PULSE_TIME
                     00017 ;
                     00018 ;
                     00019 ;       Program = COMP_1.ASM
                     00020 ;       Revision Date:   7-13-94
                     00021 ;                        1-16-97      Compatibility with MPASMWIN 1.40
                     00022 ;
                     00023 ;
                     00024 ;*****************************************************************
                     00025 ;
                     00026 ;
                     00027 ; HARDWARE SETUP
                     00028 ;    PORTB        -  Data to serial transmit on CCP pin
                     00029 ;
                     00030 ;
                     00031             INCLUDE <p16c74.inc>
                     00001         LIST
                     00002 ;P16C74.INC Standard Header File, Version 1.00 Microchip Technology
                     00318         LIST
                     00032 .
  00000000           00033 FALSE       EQU     0
  00000001           00034 TRUE        EQU     1
                     00035
                     00036             INCLUDE <COMP.h>
                     00052         list
                     00053
                     00037 ;
  00000001           00038 PICMaster   EQU     TRUE        ; A Debugging Flag
  00000001           00039 Debug       EQU     TRUE        ; A Debugging Flag
  00000001           00040 Debug_PU    EQU     TRUE        ; A Debugging Flag
                     00041 ;
                     00042 ;
                     00043 ; Reset address. Determine type of RESET
                     00044 ;
0000                 00045         org     RESET_V             ; RESET vector location
0000 1683            00046 RESET       BSF     STATUS, RP0     ; Bank 1
0001 188E            00047             BTFSC   PCON,NOT_POR    ; Power-up reset?
0002 287C            00048             GOTO    START           ; YES
0003 28BA            00049             GOTO    OTHER_RESET     ; NO, a WDT or MCLR reset
```


© 1997 Microchip Technology Inc.

```
                00050 ;
                00051 ; This is the Periperal Interrupt routine. Need to determine the type
                00052 ; of interrupt that occurred. The following interrupts are enabled:
                00053 ;   1.  CCP Capture Occured
                00054 ;
                00055     page
0004            00056         org     ISR_V             ; Interrupt vector location
0004            00057 PER_INT_V
                00058         if ( Debug )
0004 1405       00059             bsf     PORTA, 0      ; Turn on strobe
                00060         endif                     ;
0005 1283       00061             BCF     STATUS, RP0   ; Bank 0
0006 190C       00062             BTFSC   PIR1, CCP1IF  ; Compare Interrupt occured?
0007 280E       00063             GOTO    CCP1_INT      ; YES, Service the TMR1 Interrupt
0008            00064 ERROR1                            ; NO, Error Condition-Unknown Interrupt
0008 1505       00065             BSF     PORTA, 2      ; Toggle a PORT pin
0009 1105       00066             BCF     PORTA, 2
000A 2808       00067             GOTO    ERROR1
                00068 ;
000B            00069 ERROR2                            ; NO, Error Condition-Unknown Interrupt
000B 1585       00070             BSF     PORTA, 3      ; Toggle a PORT pin
000C 1185       00071             BCF     PORTA, 3
000D 280B       00072             GOTO    ERROR2
                00073 ;
                00074 ;
                00075 ;*****************************************************************
                00076 ; In the CCP interrupt.
                00077 ; Since timer1 is not cleared on a CCP match, the value in the
                00078 ; CCPR1H:CCPR1L register pair must be updated. This is done with
                00079 ; a 16-bit add. Also after the 1st CCP1 match (CCP1 pin goes high) the
                00080 ; next match will force it low. Depending on the value of the data bit
                00081 ; determines the value add to the CCPR1H:CCPR1L register pair.
                00082 ;
                00083 ; After the data has been transmitted, the pin will have a sync pulse
                00084 ; and then remain low for 300 us.
                00085 ;*****************************************************************
                00086 ;
                00087
000E            00088 CCP1_INT
000E 110C       00089             BCF     PIR1, CCP1IF    ; Clear CCP1 Interrupt Flag
000F 0AB3       00090             INCF    CCP1_INT_CNT, F ;
0010 1C33       00091             BTFSS   CCP1_INT_CNT, 0 ;
0011 2833       00092             GOTO    SYNC_PULSE
0012            00093 DATA_PULSE
0012 03B1       00094             DECF    DATA_CNT, F     ; Decrement the Count of data bits
0013 1903       00095             BTFSC   STATUS, Z       ; Have we transmitted all the Data Bits?
0014 2827       00096             GOTO    PERIOD_DELTA    ; YES, Delay to 300 us
0015 0DB0       00097             RLF     XMIT_DATA, F    ; NO, get next bit to transmit
0016 1803       00098             BTFSC   STATUS, C       ; Is the bit to transmit a '1'?
0017 281F       00099             GOTO    ONE_DATA        ; YES, Stay low for 17.6 us
0018            00100 ZERO_DATA
0018 302F       00101             MOVLW   LOW ( T_ZERO_BIT ) ; NO, Stay low for 8.8 us
0019 0795       00102             ADDWF   CCPR1L, F          ; Update Compare register pair latch
001A 1803       00103             BTFSC   STATUS, C          ;
001B 0A96       00104             INCF    CCPR1H, F          ;
001C 3000       00105             MOVLW   HIGH (T_ZERO_BIT ) ;
001D 0796       00106             ADDWF   CCPR1H, F          ;
001E 287A       00107             GOTO    RET_FIE
                00108 ;
001F            00109 ONE_DATA
001F 305E       00110             MOVLW   LOW ( T_ONE_BIT )  ; Stay low for 17.6 us
0020 0795       00111             ADDWF   CCPR1L, F          ; Update Compare register pair latch
0021 1803       00112             BTFSC   STATUS, C          ;
0022 0A96       00113             INCF    CCPR1H, F          ;
0023 3000       00114             MOVLW   HIGH ( T_ONE_BIT ) ;
0024 0796       00115             ADDWF   CCPR1H, F          ;
```

```
0025 0AB2      00116             INCF    ONES_CNT, F     ; Increment the number of 1's in the byte
0026 287A      00117             GOTO    RET_FIE
               00118 ;
0027           00119 PERIOD_DELTA
0027 0832      00120             MOVF    ONES_CNT, W     ;
0028 390F      00121             ANDLW   0x0F            ; Only want 9 states (0 1s to 8 1s)
0029 0782      00122             ADDWF   PCL, F          ;
002A 283B      00123             GOTO    ZERO_1          ; There was 0 ones in the data byte
002B 2842      00124             GOTO    ONE_1           ; There was 1 one in the data byte
002C 2849      00125             GOTO    TWO_1           ; There was 2 ones in the data byte
002D 2850      00126             GOTO    THREE_1         ; There was 3 ones in the data byte
002E 2857      00127             GOTO    FOUR_1          ; There was 4 ones in the data byte
002F 285E      00128             GOTO    FIVE_1          ; There was 5 ones in the data byte
0030 2865      00129             GOTO    SIX_1           ; There was 6 ones in the data byte
0031 286C      00130             GOTO    SEVEN_1         ; There was 7 ones in the data byte
0032 2873      00131             GOTO    EIGHT_1         ; There was 8 ones in the data byte
               00132 ;
0033           00133 SYNC_PULSE
0033 302F      00134             MOVLW   LOW ( PULSE_TIME ) ;Update Compare register pair latch
0034 0795      00135             ADDWF   CCPR1L, F          ;
0035 1803      00136             BTFSC   STATUS, C          ;
0036 0A96      00137             INCF    CCPR1H, F          ;
0037 3000      00138             MOVLW   HIGH ( PULSE_TIME );
0038 0796      00139             ADDWF   CCPR1H, F          ;
0039 1417      00140             BSF     CCP1CON, 0         ; On Compare match, CCP1 pin = L
003A 0009      00141             RETFIE
               00142 ;
003B           00143 ZERO_1
003B 30EC      00144             MOVLW   LOW ( ZERO_1S )    ; Update Compare register pair latch
003C 0795      00145             ADDWF   CCPR1L, F          ;
003D 1803      00146             BTFSC   STATUS, C          ;
003E 0A96      00147             INCF    CCPR1H, F          ;
003F 3002      00148             MOVLW   HIGH ( ZERO_1S )   ;
0040 0796      00149             ADDWF   CCPR1H, F          ;
0041 287A      00150             GOTO    RET_FIE
               00151 ;
0042           00152 ONE_1
0042 30BD      00153             MOVLW   LOW ( ONE_1S )     ; Update Compare register pair latch
0043 0795      00154             ADDWF   CCPR1L, F          ;
0044 1803      00155             BTFSC   STATUS, C          ;
0045 0A96      00156             INCF    CCPR1H, F          ;
0046 3002      00157             MOVLW   HIGH ( ONE_1S )    ;
0047 0796      00158             ADDWF   CCPR1H, F          ;
0048 287A      00159             GOTO    RET_FIE
               00160 ;
0049           00161 TWO_1
0049 308E      00162             MOVLW   LOW ( TWO_1S )     ; Update Compare register pair latch
004A 0795      00163             ADDWF   CCPR1L, F          ;
004B 1803      00164             BTFSC   STATUS, C          ;
004C 0A96      00165             INCF    CCPR1H, F          ;
004D 3002      00166             MOVLW   HIGH ( TWO_1S )    ;
004E 0796      00167             ADDWF   CCPR1H, F          ;
004F 287A      00168             GOTO    RET_FIE
               00169 ;
0050           00170 THREE_1
0050 305F      00171             MOVLW   LOW ( THREE_1S )   ; Update Compare register pair latch
0051 0795      00172             ADDWF   CCPR1L, F          ;
0052 1803      00173             BTFSC   STATUS, C          ;
0053 0A96      00174             INCF    CCPR1H, F          ;
0054 3002      00175             MOVLW   HIGH ( THREE_1S )  ;
0055 0796      00176             ADDWF   CCPR1H, F          ;
0056 287A      00177             GOTO    RET_FIE
               00178 ;
0057           00179 FOUR_1
0057 3030      00180             MOVLW   LOW ( FOUR_1S )    ; Update Compare register pair latch
0058 0795      00181             ADDWF   CCPR1L, F          ;
```


© 1997 Microchip Technology Inc.

```
0059 1803       00182             BTFSC   STATUS, C       ;
005A 0A96       00183             INCF    CCPR1H, F       ;
005B 3002       00184             MOVLW   HIGH ( FOUR_1S );
005C 0796       00185             ADDWF   CCPR1H, F       ;
005D 287A       00186             GOTO    RET_FIE
                00187 ;
005E            00188 FIVE_1
005E 3001       00189             MOVLW   LOW ( FIVE_1S ) ; Update Compare register pair latch
005F 0795       00190             ADDWF   CCPR1L, F       ;
0060 1803       00191             BTFSC   STATUS, C       ;
0061 0A96       00192             INCF    CCPR1H, F       ;
0062 3002       00193             MOVLW   HIGH ( FIVE_1S  ;
0063 0796       00194             ADDWF   CCPR1H, F       ;
0064 287A       00195             GOTO    RET_FIE
                00196 ;
0065            00197 SIX_1
0065 30D2       00198             MOVLW   LOW ( SIX_1S )  ; Update Compare register pair latch
0066 0795       00199             ADDWF   CCPR1L, F       ;
0067 1803       00200             BTFSC   STATUS, C       ;
0068 0A96       00201             INCF    CCPR1H, F       ;
0069 3001       00202             MOVLW   HIGH ( SIX_1S ) ;
006A 0796       00203             ADDWF   CCPR1H, F       ;
006B 287A       00204             GOTO    RET_FIE
                00205 ;
006C            00206 SEVEN_1
006C 30A3       00207             MOVLW   LOW ( SEVEN_1S )  ; Update Compare register pair latch
006D 0795       00208             ADDWF   CCPR1L, F         ;
006E 1803       00209             BTFSC   STATUS, C         ;
006F 0A96       00210             INCF    CCPR1H, F         ;
0070 3001       00211             MOVLW   HIGH ( SEVEN_1S ) ;
0071 0796       00212             ADDWF   CCPR1H, F         ;
0072 287A       00213             GOTO    RET_FIE
                00214 ;
0073            00215 EIGHT_1
0073 3074       00216             MOVLW   LOW ( EIGHT_1S )  ; Update Compare register pair latch
0074 0795       00217             ADDWF   CCPR1L, F         ;
0075 1803       00218             BTFSC   STATUS, C         ;
0076 0A96       00219             INCF    CCPR1H, F         ;
0077 3001       00220             MOVLW   HIGH ( EIGHT_1S ) ;
0078 0796       00221             ADDWF   CCPR1H, F         ;
0079 287A       00222             GOTO    RET_FIE
                00223
007A            00224 RET_FIE
007A 1017       00225             BCF     CCP1CON, 0        ; On Compare match, CCP1 pin = H
007B 0009       00226             RETFIE                    ; Return / Enable Global Interrupts
                00227 ;
                00228     page
                00229 ;
                00230 ;*************************************************************************
                00231 ;*****       Start program here, Power-On Reset occurred.
                00232 ;*************************************************************************
                00233 ;
007C            00234 START                                 ; POWER_ON Reset (Beginning of program)
007C 1283       00235             BCF     STATUS, RP0       ; Bank 0
007D 018F       00236             CLRF    TMR1H             ;
007E 018E       00237             CLRF    TMR1L             ;
                00238 ;
007F            00239 MCLR_RESET                            ; A Master Clear Reset
007F 1283       00240             BCF     STATUS, RP0       ; Bank 0
0080 0183       00241             CLRF    STATUS            ; Do initialization (Bank 0)
0081 018B       00242             CLRF    INTCON
0082 018C       00243             CLRF    PIR1
0083 1683       00244             BSF     STATUS, RP0       ; Bank 1
0084 3080       00245             MOVLW   0x80              ; Disable PORTB weak pull-ups
0085 0081       00246             MOVWF   OPTION_REG        ;
0086 018C       00247             CLRF    PIE1              ; Disable all peripheral interrupts
```

```
0087 30FF       00248             MOVLW   0xFF            ;
0088 009F       00249             MOVWF   ADCON1          ; Port A is Digital.
                00250 ;
                00251 ;
0089 1283       00252             BCF     STATUS, RP0     ; Bank 0
008A 0185       00253             CLRF    PORTA           ; ALL PORT output should output Low.
008B 0186       00254             CLRF    PORTB
008C 0187       00255             CLRF    PORTC
008D 0188       00256             CLRF    PORTD
008E 0189       00257             CLRF    PORTE
008F 1010       00258             BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
                00259 ;
0090 1683       00260             BSF     STATUS, RP0     ; Select Bank 1
0091 0185       00261             CLRF    TRISA           ; RA5 -  0 outputs
0092 30FF       00262             MOVLW   0xFF            ;
0093 0086       00263             MOVWF   TRISB           ; RB Port are inputs
0094 0187       00264             CLRF    TRISC           ; RC Port are outputs
0095 0188       00265             CLRF    TRISD           ; RD Port are outputs
0096 0189       00266             CLRF    TRISE           ; RE Port are outputs
0097 150C       00267             BSF     PIE1, CCP1IE    ; Enable CCP1 Interrupt
0098 1283       00268             BCF     STATUS, RP0     ; Select Bank 0
                00269 ;
                00270     page
                00271 ;
                00272 ;
                00273 ; Initialize the Special Function Registers (SFR) interrupts
                00274 ;
0099 018C       00275             CLRF    PIR1            ;
009A 0190       00276             CLRF    T1CON           ; Timer mode
009B 170B       00277             BSF     INTCON, PEIE    ; Enable Peripheral Interrupts
009C 178B       00278             BSF     INTCON, GIE     ; Enable all Interrupts
                00279 ;
                00280 ; Set-up timer and compare latches and then turn timer1 on.
                00281 ;
009D 1010       00282             BCF     T1CON, TMR1ON   ; Turn OFF timer1
009E 3041       00283             MOVLW   CCPREG_HI       ; TMR1 = CCPR1H:CCPR1L - 1
009F 008F       00284             MOVWF   TMR1H           ;
00A0 3042       00285             MOVLW   CCPREG_LO       ;
00A1 008E       00286             MOVWF   TMR1L           ;
00A2 038E       00287             DECF    TMR1L, F        ;
00A3 1803       00288             BTFSC   STATUS, C       ;
00A4 038F       00289             DECF    TMR1H, F        ;
00A5 3008       00290             MOVLW   0x08            ; On match CCP1 = H level
00A6 0097       00291             MOVWF   CCP1CON         ;
00A7 3009       00292             MOVLW   0x09            ;
00A8 00B1       00293             MOVWF   DATA_CNT        ; 8-bits to transfer
00A9 01B2       00294             CLRF    ONES_CNT        ; Result after xmit holds the number
00AA 30FF       00295             MOVLW   0xFF            ; of 1's in a byte
00AB 00B3       00296             MOVWF   CCP1_INT_CNT    ; No CCP1 transmit interrups yet
00AC 1410       00297             BSF     T1CON, TMR1ON   ; Turn ON timer1
                00298 ;
                00299 ;
                00300 ; This code segment is an infinite loop that will always transmit the data
                00301 ; contained in the XMIT_DATA register. After each byte is transmitted a
                00302 ; new byte is read. If using PICMASTER (in stand alone mode), this is
                00303 ; read from a register that is updated after a break (at NOP). If in a
                00304 ; system, PORTB is read. All other variables are reinitalized after each
                00305 ; byte.
00AD            00306 NEXT_BYTE
                00307 ;
00AD 0831       00308 WAIT        MOVF    DATA_CNT, w     ;
00AE 1D03       00309             BTFSS   STATUS, Z       ; Is DATA_CNT = 0 ?
00AF 28AD       00310             GOTO    WAIT            ; NO, must wait until YES
00B0 0000       00311             NOP                     ;
                00312         if ( Debug )
00B1 1005       00313             bcf     PORTA, 0        ; Turn off strobe
```


© 1997 Microchip Technology Inc.

```
                00314          endif                        ;
                00315
                00316          if ( PICMaster )
00B2 0840       00317              MOVF    DUMMY_PB, W      ;
                00318          else
                00319              MOVF    PORTB, W         ;
                00320          endif
00B3 00B0       00321              MOVWF   XMIT_DATA        ; New data to transmit
00B4 30FF       00322              MOVLW   0xFF             ;
00B5 00B3       00323              MOVWF   CCP1_INT_CNT     ;
00B6 3009       00324              MOVLW   0x09             ;
00B7 00B1       00325              MOVWF   DATA_CNT         ;
00B8 01B2       00326              CLRF    ONES_CNT         ;
00B9 28AD       00327              GOTO    NEXT_BYTE        ;
                00328 ;
                00329 ;
                00330 ; Here is where you do things depending on the type of RESET (Not a
                00331 ; Power-On Reset).
00BA 1E03       00332 OTHER_RESET   BTFSS  STATUS,NOT_TO ; WDT Time-out?
00BB 2808       00333 WDT_TIMEOUT   GOTO   ERROR1        ; YES, This is error condition
                00334          if ( Debug_PU )
00BC 287C       00335              goto   START          ; MCLR reset, Goto START
                00336          else
                00337              GOTO   MCLR_RESET     ; MCLR reset, Goto MCLR_RESET
                00338          endif
                00339 ;
                00340          if (Debug )
00BD 0000       00341 END_START     NOP                  ; END label for debug
                00342          endif
                00343 ;
                00344 ;
07FF            00345      org      PMEM_END             ; End of Program Memory
07FF 2808       00346               GOTO    ERROR1       ; If you get here your program was lost
                00347
                00348      end
0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXX--
07C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   191
Program Memory Words Free:  3905
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,    10 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX D:

```
    nolist
;*************************************************************************
;
;  This is the custom Header File for the real time clock application note
;   PROGRAM:CLOCK.H
;   Revision:7-13-94
;
;*************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq     EQU D'10000000'; Device Frequency is 10 MHz
PULSE_TIME   EQU (( Dev_Freq / D'4000' ) * D'188' / D'10000' )
T_ZERO_BIT   EQU (( Dev_Freq / D'4000' ) * D'188' / D'10000' )
T_ONE_BIT    EQU (( Dev_Freq / D'4000' ) * D'376' / D'10000' )
;
ZERO_1S      EQU ((( Dev_Freq / D'4000' ) * (D'6000' - (D'16' * D'188'))) / D'10000' )
ONE_1S       EQU ( ( Dev_Freq / D'4000' ) * (D'6000' - (3 * D'188' + D'14' * D'188')) / D'10000')
TWO_1S       EQU ( ( Dev_Freq / D'4000' ) * (D'6000' - (6 * D'188' + D'12' * D'188')) / D'10000')
THREE_1S     EQU ( ( Dev_Freq / D'4000' ) * (D'6000' - (D'9' * D'188' + D'10' * D'188')) / D'10000')
FOUR_1S      EQU ( ( Dev_Freq / D'4000' ) * (D'6000' - (D'12' * D'188' + 8 * D'188')) / D'10000')
FIVE_1S      EQU ( ( Dev_Freq / D'4000' ) * (D'6000' - (D'15' * D'188' + 6 * D'188')) / D'10000')
SIX_1S       EQU ( ( Dev_Freq / D'4000' ) * (D'6000' - (D'18' * D'188' + 4 * D'188')) / D'10000')
SEVEN_1S     EQU ( ( Dev_Freq / D'4000' ) * (D'6000' - (D'21' * D'188' + 2 * D'188')) / D'10000')
EIGHT_1S     EQU ( ( Dev_Freq / D'4000' ) * (D'6000' - (D'24' * D'188') ) / D'10000' )


DB_HI_BYTE   EQU (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY EQU (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR   EQU  40                ; RAM Location
OUTER_CNTR   EQU  41                ; RAM Location
;
T1OSO        EQU  0                 ; The RC0 / T1OSO / T1CKI
;
RESET_V      EQU  0x0000            ; Address of RESET Vector
ISR_V        EQU  0x0004            ; Address of Interrupt Vector
PMEM_END     EQU  0x07FF            ; Last address in Program Memory
TABLE_ADDR   EQU  0x0400            ; Address where to start Tables
;
COUNTER          EQU     0x021      ;
;
XMIT_DATA        EQU     0x30
DATA_CNT         EQU     0x31
ONES_CNT         EQU     0x32
CCP1_INT_CNT     EQU     0x33
DUMMY_PB         EQU     0x40
CCPREG_HI        EQU     0x41
CCPREG_LO        EQU     0x42

;
     list
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX E:

```
MPASM 01.40 Released           CAPT_2.ASM   1-16-1997  17:34:47         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C74, n = 66
                     00002         ERRORLEVEL  -302
                     00003 ;
                     00004 ;*****************************************************************
                     00005 ;
                     00006 ; This program implements a real time clock using the TMR1 module of the
                     00007 ; PIC16CXXX family.
                     00008 ;
                     00009 ;       Program = CAPT_2.ASM
                     00010 ;       Revision Date:   7-19-94
                     00011 ;                        1-16-97      Compatibility with MPASMWIN 1.40
                     00012 ;
                     00013 ;
                     00014 ;*****************************************************************
                     00015 ;
                     00016 ;
                     00017 ; HARDWARE SETUP
                     00018 ;
                     00019 ;           CCP2 Compare Output
                     00020 ;           CCP1 Capture Input
                     00021 ;           CCP2  --->  CCP1
                     00022 ;
                     00023 ;
                     00024             INCLUDE <p16c74.inc>
                     00001         LIST
                     00002 ; P16C74.INC Standard Header File, Ver. 1.00 Microchip Technology, Inc.
                     00318         LIST
                     00025
  00000000           00026 FALSE           EQU     0
  00000001           00027 TRUE            EQU     1
                     00028
                     00029             INCLUDE <CAPT.h>
                     00052         list
                     00030 ;
                     00031 ;
  00000001           00032 PICMaster       EQU     TRUE        ; A Debugging Flag
  00000001           00033 Debug           EQU     TRUE        ; A Debugging Flag
  00000001           00034 Debug_PU        EQU     TRUE        ; A Debugging Flag
                     00035 ;
                     00036 ;
                     00037 ; Reset address. Determine type of RESET
                     00038 ;
0000                 00039         org     RESET_V                 ; RESET vector location
0000 1683            00040 RESET       BSF     STATUS, RP0         ; Bank 1
0001 188E            00041             BTFSC   PCON,NOT_POR        ; Power-up reset?
0002 282F            00042             GOTO    START               ; YES
0003 2861            00043             GOTO    OTHER_RESET         ; NO, a WDT or MCLR reset
                     00044 ;
                     00045 ; This is the Periperal Interrupt routine. Need to determine the type
                     00046 ; of interrupt that occurred. The following interrupts are enabled:
                     00047 ;   1.  CCP1 Capture Occurred
                     00048 ;   2.  CCP2 Compare Occurred
                     00049 ;
                     00050     page
```

```
0004            00051         org     ISR_V                 ; Interrupt vector location
0004            00052 PER_INT_V
0004 1283       00053             BCF     STATUS, RP0       ; Bank 0
0005 190C       00054             BTFSC   PIR1, CCP1IF      ; CCP1 Interrupt occurred? (Capture)
0006 281D       00055             GOTO    CAPTURE           ; YES, Service the CCP1 Interrupt
0007 180D       00056             BTFSC   PIR2, CCP2IF      ; CCP2 Interrupt occurred? (Compare)
0008 2811       00057             GOTO    COMPARE           ; YES, Service the CCP2 Interrupt
0009 180C       00058             BTFSC   PIR1, TMR1IF      ; NO, Timer 1 Overflow?
000A 282D       00059             GOTO    T1OVFL            ; YES,
000B            00060 ERROR1                                ; NO, Error Condition-Unknown Interrupt
000B 1488       00061             BSF     PORTD, 1          ; Toggle a PORT pin
000C 1088       00062             BCF     PORTD, 1
000D 280B       00063             GOTO    ERROR1
                00064 ;
000E            00065 ERROR2                                ; NO, Error Condition-Unknown Interrupt
000E 1508       00066             BSF     PORTD, 2          ; Toggle a PORT pin
000F 1108       00067             BCF     PORTD, 2
0010 280E       00068             GOTO    ERROR2
                00069 ;
                00070 ; The Compare generates a Square wave based on the value on PORTB (in
                00071 ; DUMMY_PB) and on PORTD (in DUMMY_PD). PORTB is loaded into low compare
                00072 ; latch and PORTD is loaded into the high compare latch. If the value of
                00073 ; the ports is not changed, a capture overflow condition will occur when
                00074 ; PORTD:PORTB > 7Fh. This overflow is only indicated by the time between
                00075 ; captures being much less then expected.
0011            00076 COMPARE
0011 100D       00077             BCF     PIR2, CCP2IF      ; Clear CCP2 Interrupt Flag
                00078         if ( PICMaster )
0012 0851       00079             MOVF    DUMMY_PB, W       ;
                00080         else
                00081             MOVF    PORTB, W          ;
                00082         endif
0013 079B       00083             ADDWF   CCPR2L, F         ; Update Compare register pair latch
0014 1803       00084             BTFSC   STATUS, C         ;
0015 0A9C       00085             INCF    CCPR2H, F         ;
                00086         if ( PICMaster )
0016 0853       00087             MOVF    DUMMY_PD, W       ;
                00088         else
                00089             MOVF    PORTD, W          ;
                00090         endif
0017 079C       00091             ADDWF   CCPR2H, F         ;
0018 0AB3       00092             INCF    CCP2_INT_CNT, F   ;
0019 141D       00093             BSF     CCP2CON, 0        ; On Compare match, CCP2 pin = L
001A 1C33       00094             BTFSS   CCP2_INT_CNT, 0   ;
001B 101D       00095             BCF     CCP2CON, 0        ; On Compare match, CCP2 pin = H
001C            00096 END_COMPARE
001C 0009       00097             RETFIE                    ; Return / Enable Global Interrupts
                00098
                00099 ;
                00100 ; The result of the new capture minus the old capture is stored in the
                00101 ; new capture registers (CAPT_NEW_H:CAPT_NEW_L)
                00102 ;
001D            00103 CAPTURE
001D 110C       00104       BCF     PIR1, CCP1IF      ; Clear CCP1 Interrupt Flag
001E 0815       00105       MOVF    CCPR1L, W         ; New capture value (low byte)
001F 00C1       00106       MOVWF   CAPT_NEW_L        ;
0020 0816       00107       MOVF    CCPR1H, W         ; New capture value (high byte)
0021 00C0       00108       MOVWF   CAPT_NEW_H        ;
                00109 ;
0022 0843       00110       MOVF    CAPT_OLD_L, W     ;
0023 02C1       00111       SUBWF   CAPT_NEW_L, F     ; Subtract the low bytes of the 2 captures
0024 1C03       00112       BTFSS   STATUS, C         ; Did a borrow occur?
0025 03C0       00113       DECF    CAPT_NEW_H, F     ; YES, Decrement old capture (high byte)
0026 0842       00114       MOVF    CAPT_OLD_H, W     ; New capture value (low byte)
0027 02C0       00115       SUBWF   CAPT_NEW_H, F     ; Subtract the low bytes of the 2 captures
0028 0815       00116 LOAD_OLD  MOVF    CCPR1L, W     ; New capture value (low byte)
```


© 1997 Microchip Technology Inc.

```
0029 00C3       00117       MOVWF   CAPT_OLD_L      ;
002A 0816       00118       MOVF    CCPR1H, W       ; New capture value (high byte)
002B 00C2       00119       MOVWF   CAPT_OLD_H      ;
002C            00120 END_CAPTURE
002C 0009       00121       RETFIE
                00122 ;
                00123 ;
002D            00124 T1OVFL
002D 100C       00125       BCF     PIR1, TMR1IF    ; Clear T1 Overflow Interrupt Flag
002E 0009       00126       RETFIE                  ; Return / Enable Global Interrupts
                00127 ;
                00128 ;
                00129 ;*************************************************************************
                00130 ;*****      Start program here, Power-On Reset occurred.
                00131 ;*************************************************************************
                00132 ;
002F            00133 START                         ; POWER_ON Reset (Beginning of program)
002F 1283       00134       BCF     STATUS, RP0     ; Bank 0
0030 018F       00135       CLRF    TMR1H           ;
0031 018E       00136       CLRF    TMR1L           ;
                00137 ;
0032            00138 MCLR_RESET                    ; A Master Clear Reset
0032 1283       00139       BCF     STATUS, RP0     ; Bank 0
0033 0183       00140       CLRF    STATUS          ; Do initialization (Bank 0)
0034 018B       00141       CLRF    INTCON
0035 018C       00142       CLRF    PIR1
0036 1683       00143       BSF     STATUS, RP0     ; Bank 1
0037 3000       00144       MOVLW   0x00            ; The LCD module does not like to work w/
0038 0081       00145       MOVWF   OPTION_REG      ;   weak pull-ups
0039 018C       00146       CLRF    PIE1            ; Disable all peripheral interrupts
003A 018D       00147       CLRF    PIE2            ; Disable all peripheral interrupts
003B 30FF       00148       MOVLW   0xFF            ;
003C 009F       00149       MOVWF   ADCON1          ; Port A is Digital.
                00150 ;
                00151 ;
003D 1283       00152       BCF     STATUS, RP0     ; Bank 0
003E 0185       00153       CLRF    PORTA           ; ALL PORT output should output Low.
003F 0186       00154       CLRF    PORTB
0040 0187       00155       CLRF    PORTC
0041 0188       00156       CLRF    PORTD
0042 0189       00157       CLRF    PORTE
0043 1010       00158       BCF     T1CON, TMR1ON   ; Timer 1 is NOT incrementing
                00159 ;
0044 1683       00160       BSF     STATUS, RP0     ; Select Bank 1
0045 0185       00161       CLRF    TRISA           ; RA5 -  0 outputs
0046 30FF       00162       MOVLW   0xFF            ;
0047 0086       00163       MOVWF   TRISB           ; RB7 - 0 inputs
0048 0187       00164       CLRF    TRISC           ; RC Port are outputs
0049 1507       00165       BSF     TRISC, 2        ; CCP1 is an INPUT
004A 0088       00166       MOVWF   TRISD           ; RD Port are inputs
004B 0189       00167       CLRF    TRISE           ; RE Port are outputs
004C 150C       00168       BSF     PIE1, CCP1IE    ; Enable CCP1 Interrupt
004D 140D       00169       BSF     PIE2, CCP2IE    ; Enable CCP2 Interrupt
004E 1283       00170       BCF     STATUS, RP0     ; Select Bank 0
                00171 ;
                00172 ;
                00173 ; Initialize the Special Function Registers (SFR) interrupts
                00174 ;
004F 018C       00175       CLRF    PIR1            ;
0050 018D       00176       CLRF    PIR2            ;
0051 0190       00177       CLRF    T1CON           ; Timer mode
0052 170B       00178       BSF     INTCON, PEIE    ; Enable Peripheral Interrupts
0053 178B       00179       BSF     INTCON, GIE     ; Enable all Interrupts
                00180 ;
                00181 ; Set-up timer and compare latches and then turn timer1 on.
                00182 ;
```


© 1997 Microchip Technology Inc.

```
0054 1010       00183       BCF     T1CON, TMR1ON      ; Turn OFF timer1
                00184   if ( PICMaster )
0055 0851       00185       MOVF    DUMMY_PB, W        ;
                00186   else
                00187       MOVF    PORTB, W           ;
                00188   endif
0056 079B       00189       ADDWF   CCPR2L, F          ; Update Compare register pair latch
0057 1803       00190       BTFSC   STATUS, C          ;
0058 0A9C       00191       INCF    CCPR2H, F          ;
                00192   if ( PICMaster )
0059 08D3       00193       MOVF    DUMMY_PD, F        ;
                00194   else
                00195       MOVF    PORTD, W           ;
                00196   endif
005A 079C       00197       ADDWF   CCPR2H, F          ;
005B 3008       00198       MOVLW   0x08               ; On match CCP2 = H level
005C 009D       00199       MOVWF   CCP2CON            ;
005D 3005       00200       MOVLW   0x05               ; Capture on every rising edge
005E 0097       00201       MOVWF   CCP1CON            ;
005F 1410       00202       BSF     T1CON, TMR1ON      ; Turn ON timer1
                00203 ;
                00204 ;
                00205 ;
0060 2860       00206 lzz   goto    lzz                ; Loop waiting for interrupts (for use
                00207                                  ;   with PICMASTER)
                00208 ;
                00209 ; Here is where you do things depending on the type of RESET (Not a
                00210 ; Power-On Reset).
0061 1E03       00211 OTHER_RESET   BTFSS  STATUS,NOT_TO ; WDT Time-out?
0062 280B       00212 WDT_TIMEOUT   GOTO   ERROR1        ; YES, This is error condition
                00213   if ( Debug_PU )
0063 282F       00214       goto    START              ; MCLR reset, Goto START
                00215   else
                00216       GOTO    MCLR_RESET         ; MCLR reset, Goto MCLR_RESET
                00217   endif
                00218 ;
                00219   if (Debug )
0064 0000       00220 END_START     NOP                ; END lable for debug
                00221    endif
                00222 ;
                00223 ;
07FF            00224     org     PMEM_END             ; End of Program Memory
07FF 280B       00225             GOTO    ERROR1       ; If you get here your program was lost
                00226
                00227     end
0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXX----------- ----------------
07C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   102
Program Memory Words Free:  3994
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,    13 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX F:

```
    nolist
;*************************************************************************
;
;  This is the custom Header File for the real time clock application note
;   PROGRAM:CLOCK.H
;   Revision:7-19-94
;
;*************************************************************************
; This is used for the ASSEMBLER to recalculate certain frequency
; dependant variables. The value of Dev_Freq must be changed to
; reflect the frequency that the device actually operates at.
;
Dev_Freq        EQU      D'4000000'; Device Frequency is 4 MHz
DB_HI_BYTE      EQU     (HIGH ((( Dev_Freq / 4 ) * 1 / D'1000' ) / 3 ) ) + 1
LCD_INIT_DELAY  EQU     (HIGH ((( Dev_Freq / 4 ) * D'46' / D'10000' ) / 3 ) ) + 1
INNER_CNTR      EQU     40              ; RAM Location
OUTER_CNTR      EQU     41              ; RAM Location
;
T1OSO           EQU     0               ; The RC0 / T1OSO / T1CKI
;
RESET_V         EQU     0x0000          ; Address of RESET Vector
ISR_V           EQU     0x0004          ; Address of Interrupt Vector
PMEM_END        EQU     0x07FF          ; Last address in Program Memory
TABLE_ADDR      EQU     0x0400          ; Address where to start Tables
;
COUNTER         EQU     0x021           ;
CCP2_INT_CNT    EQU     0x33
;
;
; DUMMY_PD:DUMMY_PB contain the value to be loaded into the CCP2 compare registers
; (CCPR2H:CCPR2L)
;
DUMMY_PA        EQU     0x50
DUMMY_PB        EQU     0x51
DUMMY_PC        EQU     0x52
DUMMY_PD        EQU     0x53
DUMMY_PE        EQU     0x54
;
;
; CAPT_NEW_H:CAPT_NEW_L stores the NEW captured value and the result of the
; subtraction between this capture and the previous.
;    CAPT_NEW_H:CAPT_NEW_L  =  CAPT_NEW_H:CAPT_NEW_L - CAPT_OLD_H:CAPT_OLD_L
;
; After all computations the new capture value is moved to the CAPT_OLD_H:CAPT_OLD_L
; in preperation for the next capture value.
;
CAPT_NEW_H      EQU     0x040           ;
CAPT_NEW_L      EQU     0x041           ;
CAPT_OLD_H      EQU     0x042           ;
CAPT_OLD_L      EQU     0x043           ;
;
    list
```

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN597

Implementing Ultrasonic Ranging

Author: *Robert Schreiber*
Microchip Technology Inc.

INTRODUCTION

Object ranging is essential in many types of systems. One of the most popular ranging techniques is ultrasonic ranging. Ultrasonic ranging is used in a wide variety of applications including:

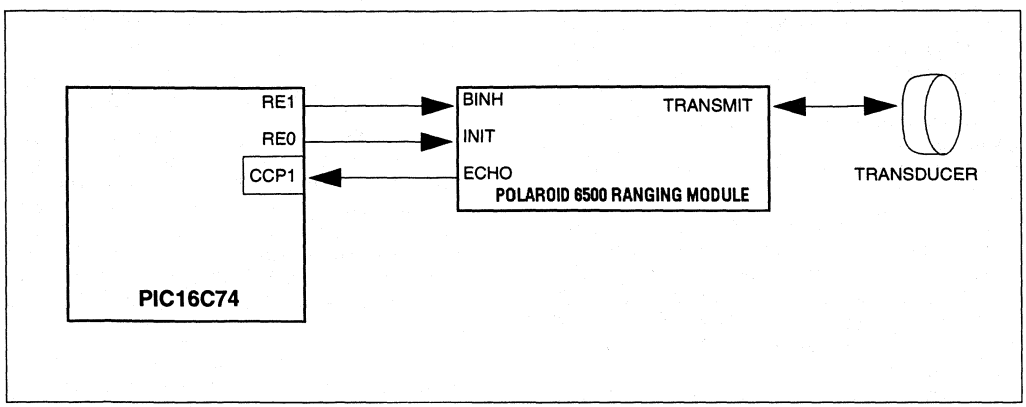
- Autofocus cameras
- Motion detection
- Robotics guidance
- Proximity sensing
- Object ranging

This application note describes a method of interfacing PIC16CXXX microcontrollers to the Polaroid 6500 Ranging Module. This implementation uses a minimum of microcontroller resources, a CCP module and two I/O pins. The two major components of the system are:

- Microcontroller
- Polaroid 6500 Ranging Module

The microcontroller performs the intelligence and arithmetic functions for ultrasonic ranging, while the Polaroid 6500 Ranging Module performs the ultrasonic signal transmissions and echo detection.

FIGURE 1: RANGING MODULE INTERFACE



THEORY OF OPERATION

Ultrasonic ranging entails transmitting a sound wave and measuring the time that it takes for the sound wave to reflect off of an object and back to the origin. The reflection time is proportional to the distance that the object is from the source. In this implementation, the sound wave is transmitted and received from the same transducer. Therefore, a blanking interval is required between signal transmission and reception to eliminate false echoes (i.e., a transmitted signal being detected as its own echo).

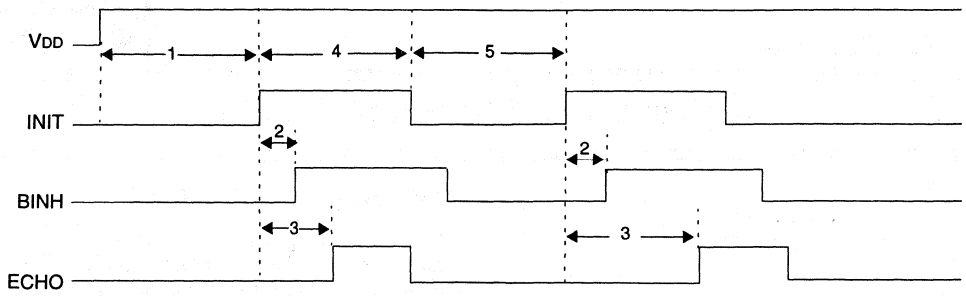
CIRCUIT CONFIGURATION

In this implementation, a PIC16C74 is connected to the ranging module as shown in Figure 1. The RE0 and RE1 I/O pins are configured as digital outputs and are tied to INIT and BINH, respectively. The CCP1 pin is configured as a digital input and is tied to ECHO through a pull-up resistor. The pull-up resistor is needed since the ECHO signal is an open-collector output. The CCP1 pin is configured for capture mode (CCP1CON). Figure 2 shows the timing relationship for V_{DD} and the three signal lines (INIT, BINH, and ECHO).

Note: The ranging module requires 5.0 milliseconds to stabilize during power-up.

© 1997 Microchip Technology Inc.

FIGURE 2: TIMING DIAGRAM OF RANGING MODULE CONTROL LINES



Parameter Number	Symbol	Characteristic	Min	Typ	Max	Units
1	TPU	Ranging Module Stabilization Time	5.0	–	–	ms
2	TBINH	Blank Inhibit Time	0.9	2.38	–	ms
3	TECHO	Echo Time	–	–	–	–
4	TINIT_H	High Time for INIT	100	–	–	ms
5	TINIT_L	Low Time for INIT	100	–	–	ms

The PIC16C74 is configured to use one of its internal timers, Timer1, in capture mode to measure the time between signal transmission and echo detection. The resolution of the timer is determined by the microcontroller clock frequency. For this application, a 4 MHz external oscillator was used, giving a resolution of 1 ms per bit. The PIC16C74 initiates a ranging cycle by first clearing Timer1. Timer1 is then enabled and INIT is immediately asserted on the ranging module. When INIT is asserted, the ranging module transmits a series of 16 pulses on the transducer at 49.4 kHz. The transmitted pulses reflect off the object and are received back at the transducer.

The transducer is used for both transmitting and receiving sound waves. A blanking interval is needed to ensure that the transmitted signal has decayed on the transducer, in order not to receive false echoes. In normal operation, the ranging module has a blanking interval of 2.38 milliseconds, which corresponds to a minimum detection distance of approximately 17 inches. However, the BINH (blank inhibit) signal can be manipulated to reduce the blanking time on the transducer to allow for object ranging as close as 6 inches.

In this implementation, the PIC16C74 asserts the BINH signal approximately 0.9 milliseconds after signal transmission. This enables the transducer to receive reflections off objects at a distance of 6 inches. The ranging module asserts the ECHO signal when a valid reflection has been detected. The PIC16C74 uses the ECHO signal to trigger a capture of the Timer1 value. The capture register contains the 16-bit value representing the elapsed time between signal transmission and echo detection. The PIC16C74 then calculates object distance based on the Timer1 value, microcontroller clock speed, and the velocity of sound in the atmosphere. The basic equation for calculating distance is given below:

Distance (inches) = TECHO time / 147.9 microseconds

Note: The minimum high and low time for INIT is 100 milliseconds, as seen in Figure 2.

DESIGN CONSIDERATIONS

There are several design considerations which must be taken into account and are listed below.

The absolute measuring distance supported by the ranging module is 6 inches to 35 feet with an accuracy of +/- 1%.

The distance output from the ranging module can be averaged over time to filter distance calculations.

In some applications, the gain of the receiver amplifier may be too low or too high and may need to be adjusted. For example, if the transducer is mounted in a cylinder, the gain may need to be lowered to reduce false echoes within the cylinder. In this case, R1 (refer to the Polariod Ultrasonic Ranging System manual) may be replaced with a 20 kΩ potentiometer to tweak the gain of the receiver amplifier to reduce false echoes.

In order for the Polaroid 6500 ranging module to operate properly, the power supply must be capable of handling high current transients (2.5 A) during the

© 1997 Microchip Technology Inc.

transmit pulse. The instantaneous drain on the power supply can be mitigated by installing a storage capacitor across the power lines at the ranging module. A value of 500 microfarads is recommended.

A 200 millisecond interval is recommended between ranging cycles (Figure 2) to allow the transducer to clear.

The ECHO line requires a pull-up resistor (4.7 kΩ was used in this application).

There must be a common ground between the PIC16C74 circuitry and the ranging module.

Some applications may not need the resources of the higher end PIC16CXXX devices. It is still possible to do this application using a device that does not contain a CCP module (for ECHO timing). The capture function can be implemented in firmware. The effect of a firmware implementation is that the resolution of the ECHO time would be 3 TCY cycles versus 1 TCY cycle for the CCP module. Also, the firmware implementation would not allow other tasks to be performed while the capture function was occurring.

Refer to Appendix A for general ranging module specifications.

© 1997 Microchip Technology Inc.

APPENDIX A: POLAROID MODULE SPECIFICATIONS

Note: This appendix contains general specifications from the Polaroid Ultrasonic Ranging System Manual. Please refer to the current Polaroid Ultrasonic Ranging System Manual for current information regarding ranging module design considerations.

DESIGN CONSIDERATIONS IN ULTRASONICS

Range: (with user custom designed processing electronics)

Farther

a) Use an acoustic horn to "focus" the sound (narrowing the beamwidth).
b) Use two transducers – 1 receiver and 1 transmitter – facing each other.
c) Lower the transmitting frequency (which will decrease the attenuation in air).

Closer

a) Use a shorter transmit signal (such as four cycles).
a) Use two transducers – one to transmit, one to receive (eliminates waiting for damping time).

Resolution

a) Above all, know the target and range well, and design a system with them in mind.
b) Use a higher transmit frequency.
c) Look at phase differences of a given cycle of the transmitted signal and received echo (as opposed to using and integration technique).
d) Increase the clock frequency of the timer.

Accuracy: (again, you must have a well defined target)

Temperature Compensate

a) Use a second small target, as a reference, at a known distance in the ranging path (such as a 1/4" rod several feet away), process both echoes, then normalize the second distance with respect to the first, since t1/d1 = t2/d2.
b) Incorporate a temperature sensing integrated circuit to drive a VCO to do the distance interval clocking.
c) To increase sensitivity of detection circuit change the value of C4 from 3300 pF to 1000 pF on the 6500 Series Ranging Module.

Beam Width:

Increase

a) Use an acoustic lens (to disperse the signal).
b) Decrease the transmitting frequency.
c) Use several transducers to span an area.

Decrease

a) Use an acoustic horn (to focus the sound).
b) Increase the transmitting frequency.

TABLE 1: RECOMMENDED OPERATING CONDITIONS

		Min.	Max.	Unit
Supply Voltage, V_{CC}		4.5	6.8	V
High-level input voltage, V_{IH}	BINH, INIT	2.1	—	V
Low-level input voltage, V_{IL}	BINH, INIT	—	0.6	V
ECHO and OSC output voltage		—	6.8	V
Delay time, power up to INIT high		5	—	ms
Recycle period		80	—	ms
Operating free-air temperature, T_A		0	40	°C

© 1997 Microchip Technology Inc.

TABLE 2: ELECTRICAL CHARACTERISTICS OVER RECOMMENDED RANGES OF SUPPLY VOLTAGE AND OPERATING FREE-AIR TEMPERATURE (UNLESS OTHERWISE NOTED)

Parameter		Test Conditions	Min.	Typ.	Max.	Unit
Input current	BINH, INIT	V1 = 2.1V	—	—	1	mA
High-level output current, I_{OH}	ECHO, OSC	V_{OH} = 5.5V	—	—	100	µA
Low-level output voltage, V_{OL}	ECHO, OSC	I_{OL} = 1.6 mA	—	—	0.4	V
Transducer bias voltage		T_A = 25°C	—	200		V
Transducer output voltage (peak-to-peak)		T_A = 25°C	—	400		V
Number of cycles for XDCR output to reach 400V		C= 500 pF	—	—	7	
Internal blanking interval			—	2.38*	—	ms
Frequency during 16-pulse transmit period	OSC output		—	49.4*	—	kHz
	XMIT output		—	49.4*	—	
Frequency after 16 pulse transmit period	OSC output		—	93.3*	—	kHz
	XMIT output		—	0	—	
Supply current, I_{CC}	During transmit period		—	—	2000	mA
	After transmit period		—	—	100	

* These typical values apply for a 420 kHz ceramic resonator.

3

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: FIRMWARE LISTING

```
MPASM 01.40 Released            XDCR.ASM   1-22-1997  10:55:26         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ; XDCR.ASM
                     00002 ;
                     00003 ; This routine continually executes ranging cycles in the
                     00004 ; following order:
                     00005 ;
                     00006 ;       1) Timers and Flags are cleared
                     00007 ;       2) Ranging Cycle Executes
                     00008 ;       3) Distance is Calculated (to 0.5 inch)
                     00009 ;       4) HW is re-initialized for next cycle
                     00010 ;
                     00011 ; The processor uses a 4MHz oscillator, so all timing
                     00012 ; calculations are referenced to that.  The calculated
                     00013 ; distance is a 16-bit result in the ACCbHI:ACCbLO registers.
                     00014 ;
                     00015 ;       Program:          XDCR.ASM
                     00016 ;       Revision Date:
                     00017 ;                         1-22-97      Compatibility with MPASMWIN 1.40
                     00018 ;
                     00019 ;
                     00020
                     00021         LIST P=16C74
                     00022 ;
                     00001         LIST
                     00002 ; P16C74.INC  Standard Header File, Version 1.00  Microchip Technology, Inc.
                     00318         LIST
                     00025
  00000030           00026 TEMP    equ     0x30            ;Temporary storage location
  00000031           00027 TEMP1   equ     0x31            ;Temporary storage location
  00000032           00028 TEMP2   equ     0x32            ;Temporary storage location
  00000033           00029 TEMP3   equ     0x33            ;Temporary storage location
  00000034           00030 COUNT1  equ     0x34            ;Temporary count register
  00000035           00031 COUNT2  equ     0x35            ;Temporary count register
                     00032 ;
                     00033 #DEFINE      XDCR               ; Flag for conditional assemble of test code
                     00034                                 ; in file DBL_DIVF.ASM. END directive MUST be
                     00035                                 ; commented out in file DBL_DIVF.ASM
                     00036 ;
                     00037 ;*****************************************************************
                     00038         LIST
                     00039
                     00040 ;*****************
                     00041 ; Bank 0 Registers
                     00042 ;*****************
                     00043 ;
                     00044 ; TMR1 is off, Prescaler is 1 for a capture timeout of 65 msec
0000 0190            00045         clrf    T1CON
                     00046 ; Set to capture on every rising edge
0001 3005            00047         movlw   0x05
0002 0097            00048         movwf   CCP1CON
                     00049 ; Clear the Ports
0003 0185            00050         clrf    PORTA
0004 0186            00051         clrf    PORTB
0005 0187            00052         clrf    PORTC
0006 0188            00053         clrf    PORTD
```


© 1997 Microchip Technology Inc.

```
0007 0189       00054         clrf    PORTE
                00055 ;
                00056 ;******************
                00057 ; Bank 1 Registers
                00058 ;******************
                00059 ;
0008 1683       00060         bsf     STATUS,RP0      ; Set Bank1
                00061 ; Port A is Digital, Port E is Digital
0009 3007       00062         movlw   0x07
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
000A 009F       00063         movwf   ADCON1
                00064 ; Configure CCP1 (RC2) as an input, and all other ports
                00065 ; as Outputs, (RE0 = INIT, RE1 = BINH)
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
000B 0185       00066         clrf    TRISA
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
000C 0186       00067         clrf    TRISB
000D 3004       00068         movlw   0x04
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
000E 0087       00069         movwf   TRISC
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
000F 0188       00070         clrf    TRISD
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0010 0189       00071         clrf    TRISE
0011 1283       00072         bcf     STATUS,RP0      ; Clear RP0
0012            00073 Xdcr
                00074 ;
                00075 ; Initialize Timers and Flags
                00076 ;
0012 1010       00077         bcf     T1CON,0 ; Disable TMR1
0013 018C       00078         clrf    PIR1    ; Clear Timer1 Overflow Flag & Timer1 Capture Flag
0014 018E       00079         clrf    TMR1L   ; Clear TMR1L
0015 018F       00080         clrf    TMR1H   ; Clear TMR1H
0016 0195       00081         clrf    CCPR1L  ; Clear CCPR1L
0017 0196       00082         clrf    CCPR1H  ; Clear CCPR1H
0018 1409       00083         bsf     PORTE,0 ; Set INIT High on Ranging Module
0019 1410       00084         bsf     T1CON,0 ; Enable TMR1
001A 21F3       00085         call    DEL_9   ; Delay 0.9 msec for transducer to stabilize
001B 1489       00086         bsf     PORTE,1 ; Enable Transducer to Receive (BINH)
001C            00087 chk_t1
001C 190C       00088         btfsc   PIR1,2          ; Check for Capture
001D 2822       00089         goto    chk_done        ; Jump if Capture
001E 1C0C       00090         btfss   PIR1,0          ; Check for TMR1 Overflow
001F 281C       00091         goto    chk_t1          ; Loop if nothing happened
0020 1010       00092         bcf     T1CON,0         ; Turn off TMR1
0021 2833       00093         goto    ovr_flo         ; Capture event did not occur
0022            00094 chk_done
                00095 ;
                00096 ; Calculate distance to 0.5 inch resolution
                00097 ;
0022 1010       00098         bcf     T1CON,0         ; Turn off TMR1
0023 0815       00099         movf    CCPR1L,W        ; Move LSB into W
0024 00A2       00100         movwf   ACCbLO          ; Move LSB into ACCbLO
0025 0816       00101         movf    CCPR1H,W        ; Move MSB into W
0026 00A3       00102         movwf   ACCbHI          ; Move MSB into ACCbHI
0027 304A       00103         movlw   0x4A            ; Move 75usec/0.50in into W
0028 00A0       00104         movwf   ACCaLO          ; Move LSB into ACCaLO
0029 01A1       00105         clrf    ACCaHI          ; Clear MSB (ACCaHI)
002A 208F       00106         call    D_divF          ; Call 16-bit/8-bit routine
                00107                                 ; which is described in
                00108                                 ; Application Note 544
002B 3025       00109         movlw   0x25            ; Check remainder to see if
002C 0224       00110         subwf   ACCcLO,W        ; we should round up...
002D 1803       00111         btfsc   STATUS,C        ; If Remainder < (0.5 * Divisor), skip
002E 0AA2       00112         incf    ACCbLO,F        ; Round up
002F 1903       00113         btfsc   STATUS,Z        ; Check low byte for wrap around
0030 0AA3       00114         incf    ACCbHI,F        ; If LSB wrapped, increment high byte
```


© 1997 Microchip Technology Inc.

```
0031 1D03       00115         btfss   STATUS,Z        ; Check high byte for wrap around
0032 2835       00116         goto    done            ; High byte didn't wrap
0033            00117 ovr_flo
0033 01A2       00118         clrf    ACCbLO
0034 01A3       00119         clrf    ACCbHI
0035            00120 done
0035 21FD       00121         call    DEL_100         ; Wait 100 msec before clearing HW.
0036 1009       00122         bcf     PORTE,0         ; Disable INIT
0037 1089       00123         bcf     PORTE,1         ; Disable BINH
0038 21FD       00124         call    DEL_100         ; Wait 100 msec before enabling HW.
0039 2812       00125         goto    Xdcr
                00126
                00319         LIST
                00320
                00636         LIST
                00044
  00000001      00045 TRUE    equ     1h
  00000000      00046 FALSE   equ     0h
                00047
0080            00048         org     0x080
                00049 ;*****************************************************************
  00000000      00050 SIGNED  equ     FALSE           ; Set This To 'TRUE' if the routines
                00051 ;                               ; for Multiplication & Division needs
                00052 ;                               ; to be assembled as Signed Integer
                00053 ;                               ; Routines. If 'FALSE' the above two
                00054 ;                               ; routines ( D_mpy & D_div ) use
                00055 ;                               ; unsigned arithmetic.
                00056 ;*****************************************************************;
                00057 ;       division macro
                00058 ;
                00059 divMac  MACRO
                00060         LOCAL   NOCHK
                00061         LOCAL   NOGO
                00062 ;
                00063         bcf     STATUS,C
                00064         rlf     ACCdLO, F
                00065         rlf     ACCdHI, F
                00066         rlf     ACCcLO, F
                00067         rlf     ACCcHI, F
                00068         movf    ACCaHI,W
                00069         subwf   ACCcHI,W        ;check if a>c
                00070         btfss   STATUS,Z
                00071         goto    NOCHK
                00072         movf    ACCaLO,W
                00073         subwf   ACCcLO,W        ;if msb equal then check lsb
                00074 NOCHK   btfss   STATUS,C        ;carry set if c>a
                00075         goto    NOGO
                00076         movf    ACCaLO,W        ;c-a into c
                00077         subwf   ACCcLO, F
                00078         btfss   STATUS,C
                00079         decf    ACCcHI, F
                00080         movf    ACCaHI,W
                00081         subwf   ACCcHI, F
                00082         bsf     STATUS,C        ;shift a 1 into b (result)
                00083 NOGO    rlf     ACCbLO, F
                00084         rlf     ACCbHI, F
                00085 ;
                00086         ENDM
                00087 ;
                00088 ;*****************************************************************
                00089 ;       Double Precision Divide ( 16/16 -> 16 )
                00090 ;
                00091 ; ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
                00092 ; with Quotiont in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc
                00093 ; (ACCcHI,ACCcLO).
                00094 ;   NOTE  :  Before calling this routine, the user should make sure that
```


© 1997 Microchip Technology Inc.

```
                00095 ;            the Numerator(ACCb) is greater than Denominator(ACCa). If
                00096 ;            the case is not true, the user should scale either Numerator
                00097 ;            or Denominator or both such that Numerator is greater than
                00098 ;            the Denominator.
                00099 ;
                00100 ;
0080 3010       00101 setup   movlw   .16             ; for 16 shifts
0081 00A8       00102         movwf   temp
0082 0823       00103         movf    ACCbHI,W       ;move ACCb to ACCd
0083 00A7       00104         movwf   ACCdHI
0084 0822       00105         movf    ACCbLO,W
0085 00A6       00106         movwf   ACCdLO
0086 01A3       00107         clrf    ACCbHI
0087 01A2       00108         clrf    ACCbLO
0088 3400       00109         retlw   0
                00110 ;
                00111 ;*****************************************************************
                00112 ;
0089 09A0       00113 neg_A   comf    ACCaLO, F      ; negate ACCa ( -ACCa -> ACCa )
008A 0AA0       00114         incf    ACCaLO, F
008B 1903       00115         btfsc   STATUS,Z
008C 03A1       00116         decf    ACCaHI, F
008D 09A1       00117         comf    ACCaHI, F
008E 3400       00118         retlw   0
                00119 ;
                00120 ;*****************************************************************
                00121
                00122 ;
008F            00123 D_divF
                00124 ;
                00125       IF   SIGNED
                00126       CALL    S_SIGN
                00127       ENDIF
                00128 ;
008F 2080       00129         call    setup
0090 01A5       00130         clrf    ACCcHI
0091 01A4       00131         clrf    ACCcLO
                00132 ;
                00133 ; use the mulMac macro 16 times
                00134 ;
                00135         divMac
  0000              M         LOCAL   NOCHK
  0000              M         LOCAL   NOGO
                    M ;
0092 1003           M         bcf     STATUS,C
0093 0DA6           M         rlf     ACCdLO, F
0094 0DA7           M         rlf     ACCdHI, F
0095 0DA4           M         rlf     ACCcLO, F
0096 0DA5           M         rlf     ACCcHI, F
0097 0821           M         movf    ACCaHI,W
0098 0225           M         subwf   ACCcHI,W       ;check if a>c
0099 1D03           M         btfss   STATUS,Z
009A 289D           M         goto    NOCHK
009B 0820           M         movf    ACCaLO,W
009C 0224           M         subwf   ACCcLO,W       ;if msb equal then check lsb
009D 1C03           M NOCHK   btfss   STATUS,C       ;carry set if c>a
009E 28A6           M         goto    NOGO
009F 0820           M         movf    ACCaLO,W       ;c-a into c
00A0 02A4           M         subwf   ACCcLO, F
00A1 1C03           M         btfss   STATUS,C
00A2 03A5           M         decf    ACCcHI, F
00A3 0821           M         movf    ACCaHI,W
00A4 02A5           M         subwf   ACCcHI, F
00A5 1403           M         bsf     STATUS,C       ;shift a 1 into b (result)
00A6 0DA2           M NOGO    rlf     ACCbLO, F
00A7 0DA3           M         rlf     ACCbHI, F
                        M ;
```

```
                    00136          divMac
  0000                  M          LOCAL   NOCHK
  0000                  M          LOCAL   NOGO
                        M ;
00A8 1003               M          bcf     STATUS,C
00A9 0DA6               M          rlf     ACCdLO, F
00AA 0DA7               M          rlf     ACCdHI, F
00AB 0DA4               M          rlf     ACCcLO, F
00AC 0DA5               M          rlf     ACCcHI, F
00AD 0821               M          movf    ACCaHI,W
00AE 0225               M          subwf   ACCcHI,W        ;check if a>c
00AF 1D03               M          btfss   STATUS,Z
00B0 28B3               M          goto    NOCHK
00B1 0820               M          movf    ACCaLO,W
00B2 0224               M          subwf   ACCcLO,W        ;if msb equal then check lsb
00B3 1C03               M NOCHK    btfss   STATUS,C        ;carry set if c>a
00B4 28BC               M          goto    NOGO
00B5 0820               M          movf    ACCaLO,W        ;c-a into c
00B6 02A4               M          subwf   ACCcLO, F
00B7 1C03               M          btfss   STATUS,C
00B8 03A5               M          decf    ACCcHI, F
00B9 0821               M          movf    ACCaHI,W
00BA 02A5               M          subwf   ACCcHI, F
00BB 1403               M          bsf     STATUS,C        ;shift a 1 into b (result)
00BC 0DA2               M NOGO     rlf     ACCbLO, F
00BD 0DA3               M          rlf     ACCbHI, F
                        M ;
                    00137          divMac
  0000                  M          LOCAL   NOCHK
  0000                  M          LOCAL   NOGO
                        M ;
00BE 1003               M          bcf     STATUS,C
00BF 0DA6               M          rlf     ACCdLO, F
00C0 0DA7               M          rlf     ACCdHI, F
00C1 0DA4               M          rlf     ACCcLO, F
00C2 0DA5               M          rlf     ACCcHI, F
00C3 0821               M          movf    ACCaHI,W
00C4 0225               M          subwf   ACCcHI,W        ;check if a>c
00C5 1D03               M          btfss   STATUS,Z
00C6 28C9               M          goto    NOCHK
00C7 0820               M          movf    ACCaLO,W
00C8 0224               M          subwf   ACCcLO,W        ;if msb equal then check lsb
00C9 1C03               M NOCHK    btfss   STATUS,C        ;carry set if c>a
00CA 28D2               M          goto    NOGO
00CB 0820               M          movf    ACCaLO,W        ;c-a into c
00CC 02A4               M          subwf   ACCcLO, F
00CD 1C03               M          btfss   STATUS,C
00CE 03A5               M          decf    ACCcHI, F
00CF 0821               M          movf    ACCaHI,W
00D0 02A5               M          subwf   ACCcHI, F
00D1 1403               M          bsf     STATUS,C        ;shift a 1 into b (result)
00D2 0DA2               M NOGO     rlf     ACCbLO, F
00D3 0DA3               M          rlf     ACCbHI, F
                        M ;
                    00138          divMac
  0000                  M          LOCAL   NOCHK
  0000                  M          LOCAL   NOGO
                        M ;
00D4 1003               M          bcf     STATUS,C
00D5 0DA6               M          rlf     ACCdLO, F
00D6 0DA7               M          rlf     ACCdHI, F
00D7 0DA4               M          rlf     ACCcLO, F
00D8 0DA5               M          rlf     ACCcHI, F
00D9 0821               M          movf    ACCaHI,W
00DA 0225               M          subwf   ACCcHI,W        ;check if a>c
00DB 1D03               M          btfss   STATUS,Z
00DC 28DF               M          goto    NOCHK
```


© 1997 Microchip Technology Inc.

```
00DD 0820               M           movf    ACCaLO,W
00DE 0224               M           subwf   ACCcLO,W          ;if msb equal then check lsb
00DF 1C03               M NOCHK     btfss   STATUS,C          ;carry set if c>a
00E0 28E8               M           goto    NOGO
00E1 0820               M           movf    ACCaLO,W          ;c-a into c
00E2 02A4               M           subwf   ACCcLO, F
00E3 1C03               M           btfss   STATUS,C
00E4 03A5               M           decf    ACCcHI, F
00E5 0821               M           movf    ACCaHI,W
00E6 02A5               M           subwf   ACCcHI, F
00E7 1403               M           bsf     STATUS,C          ;shift a 1 into b (result)
00E8 0DA2               M NOGO      rlf     ACCbLO, F
00E9 0DA3               M           rlf     ACCbHI, F
                        M ;
                     00139          divMac
  0000                  M           LOCAL   NOCHK
  0000                  M           LOCAL   NOGO
                        M ;
00EA 1003               M           bcf     STATUS,C
00EB 0DA6               M           rlf     ACCdLO, F
00EC 0DA7               M           rlf     ACCdHI, F
00ED 0DA4               M           rlf     ACCcLO, F
00EE 0DA5               M           rlf     ACCcHI, F
00EF 0821               M           movf    ACCaHI,W
00F0 0225               M           subwf   ACCcHI,W          ;check if a>c
00F1 1D03               M           btfss   STATUS,Z
00F2 28F5               M           goto    NOCHK
00F3 0820               M           movf    ACCaLO,W
00F4 0224               M           subwf   ACCcLO,W          ;if msb equal then check lsb
00F5 1C03               M NOCHK     btfss   STATUS,C          ;carry set if c>a
00F6 28FE               M           goto    NOGO
00F7 0820               M           movf    ACCaLO,W          ;c-a into c
00F8 02A4               M           subwf   ACCcLO, F
00F9 1C03               M           btfss   STATUS,C
00FA 03A5               M           decf    ACCcHI, F
00FB 0821               M           movf    ACCaHI,W
00FC 02A5               M           subwf   ACCcHI, F
00FD 1403               M           bsf     STATUS,C          ;shift a 1 into b (result)
00FE 0DA2               M NOGO      rlf     ACCbLO, F
00FF 0DA3               M           rlf     ACCbHI, F
                        M ;
                     00140          divMac
  0000                  M           LOCAL   NOCHK
  0000                  M           LOCAL   NOGO
                        M ;
0100 1003               M           bcf     STATUS,C
0101 0DA6               M           rlf     ACCdLO, F
0102 0DA7               M           rlf     ACCdHI, F
0103 0DA4               M           rlf     ACCcLO, F
0104 0DA5               M           rlf     ACCcHI, F
0105 0821               M           movf    ACCaHI,W
0106 0225               M           subwf   ACCcHI,W          ;check if a>c
0107 1D03               M           btfss   STATUS,Z
0108 290B               M           goto    NOCHK
0109 0820               M           movf    ACCaLO,W
010A 0224               M           subwf   ACCcLO,W          ;if msb equal then check lsb
010B 1C03               M NOCHK     btfss   STATUS,C          ;carry set if c>a
010C 2914               M           goto    NOGO
010D 0820               M           movf    ACCaLO,W          ;c-a into c
010E 02A4               M           subwf   ACCcLO, F
010F 1C03               M           btfss   STATUS,C
0110 03A5               M           decf    ACCcHI, F
0111 0821               M           movf    ACCaHI,W
0112 02A5               M           subwf   ACCcHI, F
0113 1403               M           bsf     STATUS,C          ;shift a 1 into b (result)
0114 0DA2               M NOGO      rlf     ACCbLO, F
0115 0DA3               M           rlf     ACCbHI, F
```


© 1997 Microchip Technology Inc.

```
                    M ;
                  00141         divMac
  0000              M           LOCAL   NOCHK
  0000              M           LOCAL   NOGO
                    M ;
0116 1003           M           bcf     STATUS,C
0117 0DA6           M           rlf     ACCdLO, F
0118 0DA7           M           rlf     ACCdHI, F
0119 0DA4           M           rlf     ACCcLO, F
011A 0DA5           M           rlf     ACCcHI, F
011B 0821           M           movf    ACCaHI,W
011C 0225           M           subwf   ACCcHI,W          ;check if a>c
011D 1D03           M           btfss   STATUS,Z
011E 2921           M           goto    NOCHK
011F 0820           M           movf    ACCaLO,W
0120 0224           M           subwf   ACCcLO,W          ;if msb equal then check lsb
0121 1C03           M NOCHK     btfss   STATUS,C          ;carry set if c>a
0122 292A           M           goto    NOGO
0123 0820           M           movf    ACCaLO,W          ;c-a into c
0124 02A4           M           subwf   ACCcLO, F
0125 1C03           M           btfss   STATUS,C
0126 03A5           M           decf    ACCcHI, F
0127 0821           M           movf    ACCaHI,W
0128 02A5           M           subwf   ACCcHI, F
0129 1403           M           bsf     STATUS,C          ;shift a 1 into b (result)
012A 0DA2           M NOGO      rlf     ACCbLO, F
012B 0DA3           M           rlf     ACCbHI, F
                    M ;
                  00142         divMac
  0000              M           LOCAL   NOCHK
  0000              M           LOCAL   NOGO
                    M ;
012C 1003           M           bcf     STATUS,C
012D 0DA6           M           rlf     ACCdLO, F
012E 0DA7           M           rlf     ACCdHI, F
012F 0DA4           M           rlf     ACCcLO, F
0130 0DA5           M           rlf     ACCcHI, F
0131 0821           M           movf    ACCaHI,W
0132 0225           M           subwf   ACCcHI,W          ;check if a>c
0133 1D03           M           btfss   STATUS,Z
0134 2937           M           goto    NOCHK
0135 0820           M           movf    ACCaLO,W
0136 0224           M           subwf   ACCcLO,W          ;if msb equal then check lsb
0137 1C03           M NOCHK     btfss   STATUS,C          ;carry set if c>a
0138 2940           M           goto    NOGO
0139 0820           M           movf    ACCaLO,W          ;c-a into c
013A 02A4           M           subwf   ACCcLO, F
013B 1C03           M           btfss   STATUS,C
013C 03A5           M           decf    ACCcHI, F
013D 0821           M           movf    ACCaHI,W
013E 02A5           M           subwf   ACCcHI, F
013F 1403           M           bsf     STATUS,C          ;shift a 1 into b (result)
0140 0DA2           M NOGO      rlf     ACCbLO, F
0141 0DA3           M           rlf     ACCbHI, F
                    M ;
                  00143         divMac
  0000              M           LOCAL   NOCHK
  0000              M           LOCAL   NOGO
                    M ;
0142 1003           M           bcf     STATUS,C
0143 0DA6           M           rlf     ACCdLO, F
0144 0DA7           M           rlf     ACCdHI, F
0145 0DA4           M           rlf     ACCcLO, F
0146 0DA5           M           rlf     ACCcHI, F
0147 0821           M           movf    ACCaHI,W
0148 0225           M           subwf   ACCcHI,W          ;check if a>c
0149 1D03           M           btfss   STATUS,Z
```


© 1997 Microchip Technology Inc.

```
014A 294D               M          goto    NOCHK
014B 0820               M          movf    ACCaLO,W
014C 0224               M          subwf   ACCcLO,W        ;if msb equal then check lsb
014D 1C03               M NOCHK    btfss   STATUS,C        ;carry set if c>a
014E 2956               M          goto    NOGO
014F 0820               M          movf    ACCaLO,W        ;c-a into c
0150 02A4               M          subwf   ACCcLO, F
0151 1C03               M          btfss   STATUS,C
0152 03A5               M          decf    ACCcHI, F
0153 0821               M          movf    ACCaHI,W
0154 02A5               M          subwf   ACCcHI, F
0155 1403               M          bsf     STATUS,C        ;shift a 1 into b (result)
0156 0DA2               M NOGO     rlf     ACCbLO, F
0157 0DA3               M          rlf     ACCbHI, F
                        M ;
                    00144          divMac
  0000                  M          LOCAL   NOCHK
  0000                  M          LOCAL   NOGO
                        M ;
0158 1003               M          bcf     STATUS,C
0159 0DA6               M          rlf     ACCdLO, F
015A 0DA7               M          rlf     ACCdHI, F
015B 0DA4               M          rlf     ACCcLO, F
015C 0DA5               M          rlf     ACCcHI, F
015D 0821               M          movf    ACCaHI,W
015E 0225               M          subwf   ACCcHI,W        ;check if a>c
015F 1D03               M          btfss   STATUS,Z
0160 2963               M          goto    NOCHK
0161 0820               M          movf    ACCaLO,W
0162 0224               M          subwf   ACCcLO,W        ;if msb equal then check lsb
0163 1C03               M NOCHK    btfss   STATUS,C        ;carry set if c>a
0164 296C               M          goto    NOGO
0165 0820               M          movf    ACCaLO,W        ;c-a into c
0166 02A4               M          subwf   ACCcLO, F
0167 1C03               M          btfss   STATUS,C
0168 03A5               M          decf    ACCcHI, F
0169 0821               M          movf    ACCaHI,W
016A 02A5               M          subwf   ACCcHI, F
016B 1403               M          bsf     STATUS,C        ;shift a 1 into b (result)
016C 0DA2               M NOGO     rlf     ACCbLO, F
016D 0DA3               M          rlf     ACCbHI, F
                        M ;
                    00145          divMac
  0000                  M          LOCAL   NOCHK
  0000                  M          LOCAL   NOGO
                        M ;
016E 1003               M          bcf     STATUS,C
016F 0DA6               M          rlf     ACCdLO, F
0170 0DA7               M          rlf     ACCdHI, F
0171 0DA4               M          rlf     ACCcLO, F
0172 0DA5               M          rlf     ACCcHI, F
0173 0821               M          movf    ACCaHI,W
0174 0225               M          subwf   ACCcHI,W        ;check if a>c
0175 1D03               M          btfss   STATUS,Z
0176 2979               M          goto    NOCHK
0177 0820               M          movf    ACCaLO,W
0178 0224               M          subwf   ACCcLO,W        ;if msb equal then check lsb
0179 1C03               M NOCHK    btfss   STATUS,C        ;carry set if c>a
017A 2982               M          goto    NOGO
017B 0820               M          movf    ACCaLO,W        ;c-a into c
017C 02A4               M          subwf   ACCcLO, F
017D 1C03               M          btfss   STATUS,C
017E 03A5               M          decf    ACCcHI, F
017F 0821               M          movf    ACCaHI,W
0180 02A5               M          subwf   ACCcHI, F
0181 1403               M          bsf     STATUS,C        ;shift a 1 into b (result)
0182 0DA2               M NOGO     rlf     ACCbLO, F
```


© 1997 Microchip Technology Inc.

```
0183 0DA3                M          rlf     ACCbHI, F
                         M ;
                     00146          divMac
  0000                   M          LOCAL   NOCHK
  0000                   M          LOCAL   NOGO
                         M ;
0184 1003                M          bcf     STATUS,C
0185 0DA6                M          rlf     ACCdLO, F
0186 0DA7                M          rlf     ACCdHI, F
0187 0DA4                M          rlf     ACCcLO, F
0188 0DA5                M          rlf     ACCcHI, F
0189 0821                M          movf    ACCaHI,W
018A 0225                M          subwf   ACCcHI,W        ;check if a>c
018B 1D03                M          btfss   STATUS,Z
018C 298F                M          goto    NOCHK
018D 0820                M          movf    ACCaLO,W
018E 0224                M          subwf   ACCcLO,W        ;if msb equal then check lsb
018F 1C03                M NOCHK    btfss   STATUS,C        ;carry set if c>a
0190 2998                M          goto    NOGO
0191 0820                M          movf    ACCaLO,W        ;c-a into c
0192 02A4                M          subwf   ACCcLO, F
0193 1C03                M          btfss   STATUS,C
0194 03A5                M          decf    ACCcHI, F
0195 0821                M          movf    ACCaHI,W
0196 02A5                M          subwf   ACCcHI, F
0197 1403                M          bsf     STATUS,C        ;shift a 1 into b (result)
0198 0DA2                M NOGO     rlf     ACCbLO, F
0199 0DA3                M          rlf     ACCbHI, F
                         M ;
                     00147          divMac
  0000                   M          LOCAL   NOCHK
  0000                   M          LOCAL   NOGO
                         M ;
019A 1003                M          bcf     STATUS,C
019B 0DA6                M          rlf
ACCdLO, F
019C 0DA7                M          rlf     ACCdHI, F
019D 0DA4                M          rlf     ACCcLO, F
019E 0DA5                M          rlf     ACCcHI, F
019F 0821                M          movf    ACCaHI,W
01A0 0225                M          subwf   ACCHI,W         ;check if a>c
01A1 1D03                M          btfss   STAUS,Z
01A2 29A5                M          goto    NOCHK
01A3 0820                M          movf    ACCaLO,W
01A4 0224                M          subwf   ACCcLO,W        ;if msb equal then check lsb
01A5 1C03                M NOCHK    btfss   STAUS,C         ;carry set if c>a
01A6 29AE                M          goto    NOGO
01A7 0820                M          movf    ACCaLO,W        ;c-a into c
01A8 02A4                M          subwf   ACCcLO, F
01A9 1C03                M          btfss   STAUS,C
01AA 03A5                M          decf    ACCHI, F
01AB 0821                M          movf    ACCaHI,W
01AC 02A5                M          subwf   ACCHI, F
01AD 1403                M          bsf     STAUS,C         ;shift a 1 into b (result)
01AE 0DA2                M NOGO     rlf     ACCbLO, F
01AF 0DA3                M          rlf     ACCbHI, F
                         M ;
                     00148          divMac
  0000                   M          LOCAL   NOCHK
  0000                   M          LOCAL   NOGO
                         M ;
01B0 1003                M          bcf     STATUS,C
01B1 0DA6                M          rlf     ACCdLO, F
01B2 0DA7                M          rlf     ACCHI, F
01B3 0DA4                M          rlf     ACCcLO, F
01B4 0DA5                M          rlf     ACCcHI, F
01B5 0821                M          movf    ACCaHI,W
```


© 1997 Microchip Technology Inc.

```
01B6 0225                M          subwf   ACCHI,W          ;check if a>c
01B7 1D03                M          btfss   STATUS,Z
01B8 29BB                M          goto    NOCHK
01B9 0820                M          movf    ACCaLO,W
01BA 0224                M          subwf   ACCcLO,W         ;if msb equal then check lsb
01BB 1C03                M NOCHK    btfss   STATUS,C         ;carry set if c>a
01BC 29C4                M          goto    NOGO
01BD 0820                M          movf    ACCaLO,W         ;c-a into c
01BE 02A4                M          subwf   ACCcLO, F
01BF 1C03                M          btfss   STATUS,C
01C0 03A5                M          decf    ACCcHI, F
01C1 0821                M          movf    ACCaHI,W
01C2 02A5                M          subwf   ACCcHI, F
01C3 1403                M          bsf     STATUS,C         ;shift a 1 into b (result)
01C4 0DA2                M NOGO     rlf     ACCbLO, F
01C5 0DA3                M          rlf     ACCbHI, F
                         M ;
                       00149        divMac
  0000                   M          LOCAL   NOCHK
  0000                   M          LOCAL   NOGO
                         M ;
01C6 1003                M          bcf     STATUS,C
01C7 0DA6                M          rlf     ACCdLO, F
01C8 0DA7                M          rlf     ACCdHI, F
01C9 0DA4                M          rlf     ACCcLO, F
01CA 0DA5                M          rlf     ACCcHI, F
01CB 0821                M          movf    ACCaHI,W
01CC 0225                M          subwf   ACCcHI,W         ;check if a>c
01CD 1D03                M          btfss   STATUS,Z
01CE 29D1                M          goto    NOCHK
01CF 0820                M          movf    ACCaLO,W
01D0 0224                M          subwf   ACCcLO,W         ;if msb equal then check lsb
01D1 1C03                M NOCHK    btfss   STATUS,C         ;carry set if c>a
01D2 29DA                M          goto    NOGO
01D3 0820                M          movf    ACCaLO,W         ;c-a into c
01D4 02A4                M          subwf   ACCcLO, F
01D5 1C03                M          btfss   STATUS,C
01D6 03A5                M          decf    ACCcHI, F
01D7 0821                M          movf    ACCaHI,W
01D8 02A5                M          subwf   ACCcHI, F
01D9 1403                M          bsf     STATUS,C         ;shift a 1 into b (result)
01DA 0DA2                M NOGO     rlf     ACCbLO, F
01DB 0DA3                M          rlf     ACCbHI, F
                         M ;
                       00150        divMac
  0000                   M          LOCAL   NOCHK
  0000                   M          LOCAL   NOGO
                         M ;
01DC 1003                M          bcf     STATUS,C
01DD 0DA6                M          rlf     ACCdLO, F
01DE 0DA7                M          rlf     ACCdHI, F
01DF 0DA4                M          rlf     ACCcLO, F
01E0 0DA5                M          rlf     ACCcHI, F
01E1 0821                M          movf    ACCaHI,W
01E2 0225                M          subwf   ACCcHI,W         ;check if a>c
01E3 1D03                M          btfss   STATUS,Z
01E4 29E7                M          goto    NOCHK
01E5 0820                M          movf    ACCaLO,W
01E6 0224                M          subwf   ACCcLO,W         ;if msb equal then check lsb
01E7 1C03                M NOCHK    btfss   STATUS,C         ;carry set if c>a
01E8 29F0                M          goto    NOGO
01E9 0820                M          movf    ACCaLO,W         ;c-a into c
01EA 02A4                M          subwf   ACCcLO, F
01EB 1C03                M          btfss   STATUS,C
01EC 03A5                M          decf    ACCcHI, F
01ED 0821                M          movf    ACCaHI,W
01EE 02A5                M          subwf   ACCcHI, F
```


© 1997 Microchip Technology Inc.

```
01EF 1403              M         bsf     STATUS,C          ;shift a 1 into b (result)
01F0 0DA2              M NOGO    rlf     ACCbLO, F
01F1 0DA3              M         rlf     ACCbHI, F
                       M ;
                   00151 ;
                   00152     IF    SIGNED
                   00153         btfss   sign,MSB          ; check sign if negative
                   00154         retlw   0
                   00155         goto    neg_B             ; negate ACCa ( -ACCa -> ACCa )
                   00156     ELSE
01F2 3400          00157         retlw   0
                   00158     ENDIF
                   00159 ;
                   00160 ;*****************************************************************
                   00161 ;  Assemble this section only if Signed Arithmetic Needed
                   00162 ;
                   00163      IF    SIGNED
                   00164 ;
                   00165 S_SIGN  movf    ACCaHI,W
                   00166         xorwf   ACCbHI,W
                   00167         movwf   sign
                   00168         btfss   ACCbHI,MSB        ; if MSB set go & negate ACCb
                   00169         goto    chek_A
                   00170 ;
                   00171         comf    ACCbLO, F         ; negate ACCb
                   00172         incf    ACCbLO, F
                   00173         btfsc   STATUS,Z
                   00174         decf    ACCbHI, F
                   00175         comf    ACCbHI, F
                   00176 ;
                   00177 chek_A  btfss   ACCaHI,MSB        ; if MSB set go & negate ACCa
                   00178         retlw   0
                   00179         goto    neg_A
                   00180 ;
                   00181      ENDIF
                   00182 ;
                   00183 ;
                   00184 ;
                   00185     ifdef XDCR
                   00186 ;
                   00187 ; This file has been included, do not have test program.
                   00188 ;
                   00189     else
                   00190 ;
                   00191 ;*****************************************************************
                   00192 ;                          Test Program
                   00193 ;*****************************************************************
                   00194 ;    Load constant values to ACCa & ACCb for testing
                   00195 ;
                   00196 main    movlw   1
                   00197         movwf   ACCaHI
                   00198         movlw   0FF               ; loads ACCa = 01FF
                   00199         movwf   ACCaLO
                   00200 ;
                   00201         movlw   07F
                   00202         movwf   ACCbHI
                   00203         movlw   0FF               ; loads ACCb = 7FFF
                   00204         movwf   ACCbLO
                   00205 ;
                   00206         call    D_divF            ; remainder in ACCc. Here ACCb =0040 &
                   00207                                   ; ACCc=003F
                   00208 self    goto    self
                   00209 ;
                   00210 ;       org     PIC54
                   00211 ;       LIST    p=16c54
                   00212 ;       goto    main
                 00213 ;*****************************************************************
```


© 1997 Microchip Technology Inc.

```
                00214 ;
                00215
                00216     endif
                00217 ;
                00218 ;     END ; END directive MUST be commented out if this file is included.
                00129       LIST
                00130
                00158       LIST
                00159
                00160       end

MPASM 01.40 Released              XDCR.ASM   1-22-1997  10:55:26         PAGE 15
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXX------
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0200 : XXXXXX---------- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   448
Program Memory Words Free:  3648


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     7 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Air Flow Control Using Fuzzy Logic

Author: *Robert Schreiber*
Microchip Technology Inc.

INTRODUCTION

Fuzzy logic control can be used to implement a wide variety of intelligent functions including everything from consumer electronic goods and household appliances to auto electronics, process control, and automation.

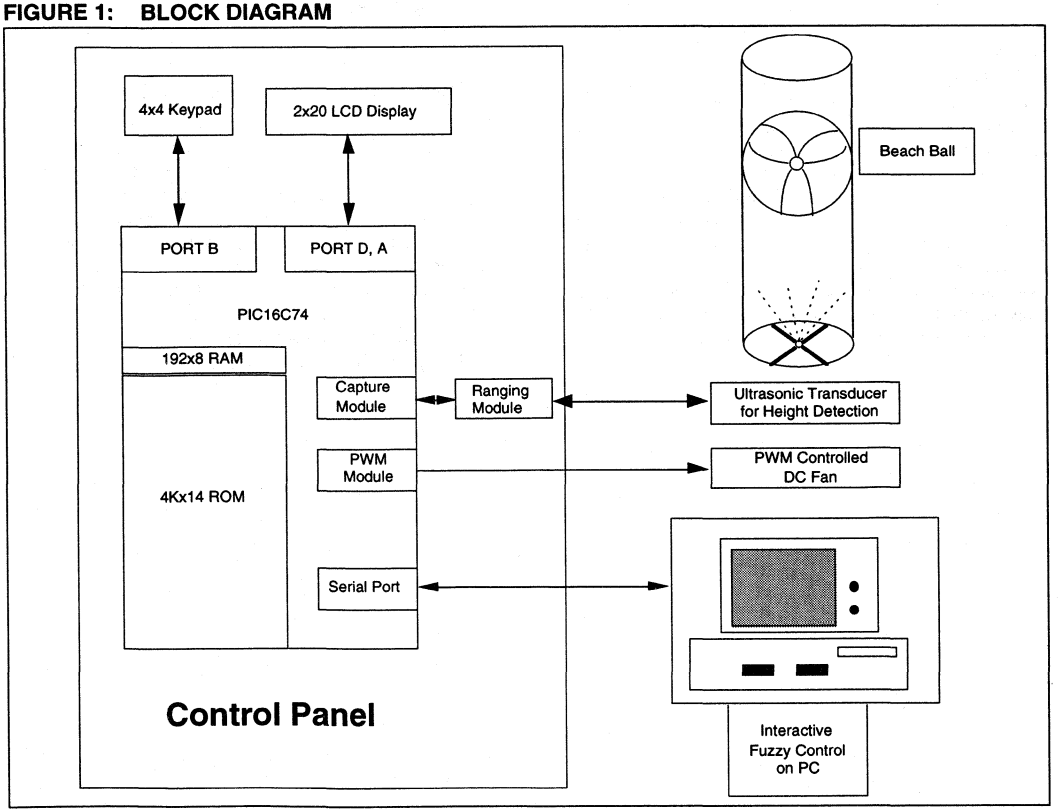
Typically, fuzzy logic control applications fall into two categories. First, it can be used to enhance existing products with intelligent functions. Second, it can utilize sensors that continuously respond to changing input conditions. In addition, fuzzy logic simplifies dealing with non-linearities in systems, and allows for quicker product development cycles.

This application note will step the user through a fuzzy logic control design utilizing sensors. The development tool used is Inform® Software's *fuzzyTECH®-MP*. The development tool allows for an all-graphical editor, analyzers, and debug capability.

PROJECT DESCRIPTION

The block diagram of the project is shown in Figure 1 and operates as follows.

FIGURE 1: BLOCK DIAGRAM



© 1997 Microchip Technology Inc.

The control panel prompts the user to enter the desired beach ball height on the 16-key keypad. The keypad input is echoed on the LCD module and the user is prompted for confirmation. Upon confirmation of user input, the control panel initiates a ranging cycle to calculate the current height of the beach ball. The desired height and current height are continually displayed on the LCD module. From the current height, the control panel calculates both the velocity and the delta height (difference in desired height from current height). This information, along with the desired height, is transmitted to the PC via an RS-232 link. The fuzzy logic algorithm, running on the PC, calculates the appropriate duty cycle of the DC fan and transmits this information to the control panel. This emulates a "real world" environment in which system level debug can be done on the PC in real-time. The control panel controls the duty cycle of the DC fan with this input. The above listed ranging process continues indefinitely until interrupted by the user.

The control panel houses an ultrasonic ranging module and the microcontroller. The microcontroller handles all of the peripheral interfaces including the 16-key keypad, the LCD display, the ultrasonic ranging module, and the RS-232 serial link. The project required a microcontroller that could handle the data throughput and all of these peripherals with little or no external components. The microcontroller used was the PIC16C74, which contains 4K of on-chip program memory and 192 bytes of on-chip data memory. Furthermore, the interrupt capabilities, I/O pins, PWM module, capture and compare modules, timer modules, Universal Asynchronous Receiver Transmitter (USART), and A/D converter make it an excellent fit for the application. In addition, the on-chip Pulse Width Modulation (PWM) module allows for a single component (FET) interface for the DC fan control and the ranging module can interface directly to the microcontroller (refer to Application Note AN597, "Implementing Ultrasonic Ranging").

FUZZY DESIGN

Fuzzy logic first translates the crisp inputs from the sensor into a linguistic description. Then it evaluates the control strategy contained in fuzzy logic rules and translates the result back into a crisp value.

The first step in fuzzy logic control design is system definition. The only possible sources of inputs to the fuzzy logic control algorithm are the ultrasonic transducer, the user, and the DC fan. The key is to decide which of these inputs are significant and which are not. Basically, the behavior of the beach ball was characterized by asking the following questions from the beach ball's perspective:

- Where am I?
- How far am I from where I want to be?
- How fast am I getting there?
- What external force will get me there?

The nice thing about fuzzy logic control is that the linguistic system definition becomes the control algorithm.

The variables were defined as follows:

- **Current Height** [Where am I?]
- **Delta Height** [How far am I from where I want to be?]
- **Velocity** [How fast am I getting there?)
- **Duty Cycle** [What external force will get me there?]

Defining the variables was the starting point, but for the algorithm to work smoothly, it isn't good enough to say "the beach ball has velocity," you need to know to what degree the beach ball has velocity. This is accomplished by defining terms that more fully describe the variable. The combination of variables and terms gives a linguistic description of what is happening to the system. From this, the Velocity variable can be described as having a "positive small velocity" or a "positive big velocity," not just a "velocity."

© 1997 Microchip Technology Inc.

There is no fixed rule on how many terms to define per variable. Typically, three to five terms are defined, but more or less may be needed based on the control algorithm. In retrospect, we probably could have reduced Current Height to three terms and Velocity to five terms. Table 1 lists the four variables that are used for the trade show demo and their associated terms.

Once the linguistic variables are defined, data types and values need to be defined. For this application, data types were defined as 8-bit integers (16-bit definition is also possible). After defining the data types, the shell and code values for each variable were specified. A shell value is used within the fuzzy logic development tool and a code value is used when the code is generated.

The best way to describe shell and code values is using the analogy of a D/A converter. If we have a 5.0V, 8-bit D/A converter, the digital input would correspond to the code value and the analog output would correspond to the shell value. This is, if we write (or pass) a value of 128 to the D/A we would get a 2.51V out. Applying this analogy to our project, we would pass a crisp value (digital) to the fuzzy world and the fuzzy world would use the fuzzy value (analog).

Therefore, when we define shell and code values, we are basically defining the "D/A converter." For example, you can define the shell value for Duty Cycle to be a minimum of 0 and a maximum of 100 (percent). Therefore, within the fuzzy logic development tool, Duty Cycle will take on a value between 0 and 100, inclusive.

The code value is limited by the data type, but can take on any or all of the digital range. That is, if the shell value is 0 to 100, the code values could be defined as 0 to 100. But to get full resolution, the code value should be defined over the entire range (i.e., 0 to 255 for 8-bit data types). The code values and shell values were defined as shown in Table 2. Note that for the height and velocity variables, the shell values are scaled by 2 (i.e., a Current Height with a crisp value of 60 would correspond to 30 inches).

TABLE 1: INPUT AND OUTPUT VARIABLES AND TERMS

Input Variables			Output Variable
Current Height	**Delta Height**	**Velocity**	**Duty Cycle**
very lo	neg big	neg big	very slo
lo	neg small	neg med	slo
medium	zero	neg small	medium slo
hi	pos small	zero	medium
very hi	pos big	pos small	medium fast
		pos med	fast
		pos big	very fast

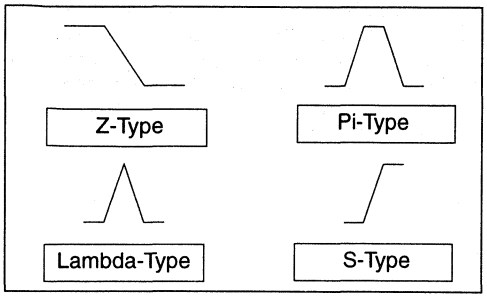
TABLE 2: SHELL AND CODE VALUES

Shell Value			Code Value	
Variable	**Min.**	**Max.**	**Min.**	**Max.**
Current Height	0	120	0	255
Delta Height	-50	50	0	255
Velocity	-5	5	0	255
Duty Cycle	0	255	0	255

© 1997 Microchip Technology Inc.

Next, the membership functions were defined to further describe the variables. The fuzzy logic development tool creates the membership functions automatically. This gives a good starting point, but the membership functions still need to be fine-tuned during the debug phase. In this application, only the linear shaped functions (Pi, Z, S and Lambda types) were used as seen in Figure 2.

FIGURE 2: STANDARD MEMBERSHIP FUNCTION TYPES



FUZZIFICATION

Fuzzification entails translating a crisp value into a fuzzy value. Once all of the variables have been defined, the interfaces between the variables need to be defined. The interfaces for the input variables contain the fuzzification procedures. In defining the interfaces, the input variable's fuzzification method needs to be defined. The computation of fuzzification is carried out at runtime for code efficiency. The type of fuzzification used in this project is membership function computation. This is largely due to the code space efficiency and accuracy associated with this method. Once fuzzification has taken place, the algorithm is performed in the fuzzy world according to the rule base.

FUZZY RULE BASE

The entire fuzzy inference is contained within the rule blocks of a system. For example, if the beach ball is near the top of the tube and it was commanded to be near the bottom of the tube, the rule that described the situation would be:

IF CURRENT HEIGHT = VERY HI

AND DELTA HEIGHT = NEGATIVE BIG

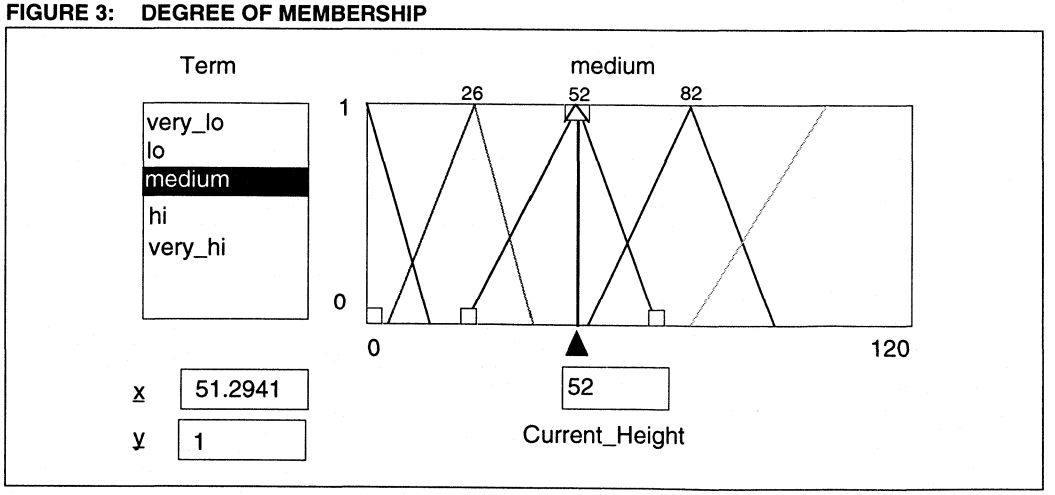
THEN DUTY CYCLE = SLOW

The above rule describes one situation, but the rule definition would continue until the system was adequately described The rule block is the collection of all rules that describe the system.

The rules of the rule block can also be defined in terms of how much a specific rule is supported when calculating inference. The support of a rule, or plausibility, is known as the degree of support for that rule. A plausible rule is defined by a 1.0, a totally implausible rule is defined by 0.0. In this project all rules are fully supported.

The degree to which a crisp value belongs to a term is known as the degree of membership. For example, the terms Medium and Hi for the variable Current Height were defined as a Lambda-type membership function centered around the crisp values 52 (26 inches) and 82 (41 inches), respectively, as shown in Figure 3.

FIGURE 3: DEGREE OF MEMBERSHIP



© 1997 Microchip Technology Inc.

Therefore, if the beach ball was at 26 inches, the degree of membership would be 1.0 for Medium and 0.0 for Hi. However, as the beach ball rises in height, the degree of membership for the term Medium would decrease and the degree of membership for the term Hi would increase. The interplay of these linguistic variable terms is controlled by the rule base. The rule base defines not only the relationship between the terms, but also how much each rule is supported, as described previously.

From the list of rules, a Fuzzy Associative Map (FAM) is constructed (see below). The FAM shows the plausibility (degree of support) of each rule as seen in Figure 4 and Figure 5.

FIGURE 4: MATRIX RULE EDITOR WITH FAM RULES

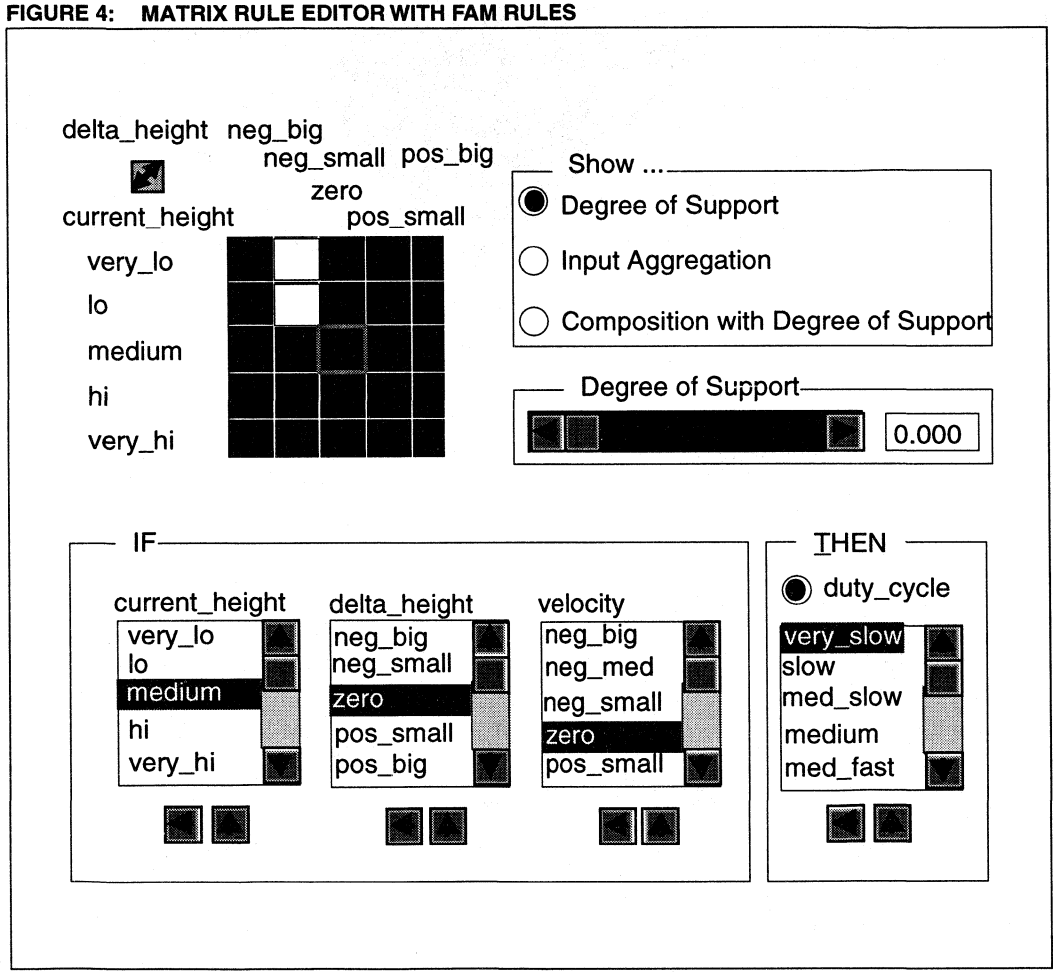
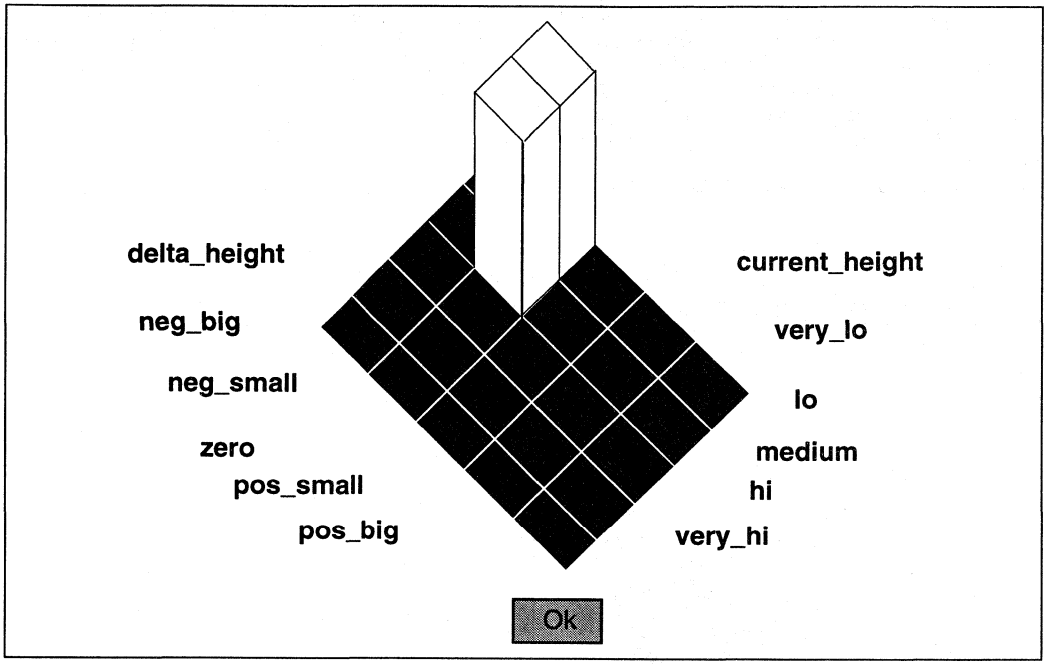


FIGURE 5: 3-D RULE DISPLAY



© 1997 Microchip Technology Inc.

DEFUZZIFICATION

Defuzzification entails translating a fuzzy value to a crisp value. The interface for the output variables contains the defuzzification procedures. For most control applications (and this project), the center-of-maximum (CoM) method is used for defuzzification. CoM evaluates more than one output term as valid and compromises between them by computing a weighted mean of the term membership maxima. Example 1 and Figure 6 show the defuzzification of the linguistic variable Duty Cycle using CoM.

EXAMPLE 1: DEFUZZIFICATION OF DUTY CYCLE

The crisp values of the three input variables are as follows:

Current Height: 30

Delta Height: 0

Velocity: 0

The crisp value can be calculated using the CoM method with the following equation.

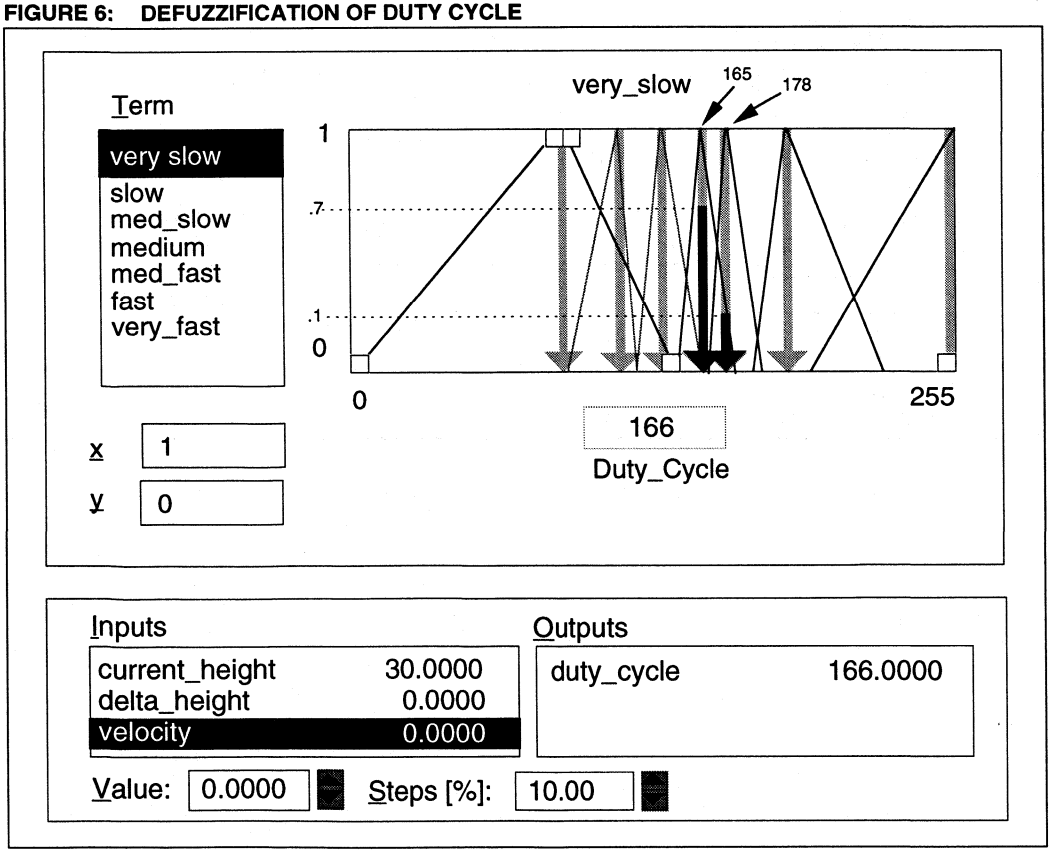
$$C = \frac{\sum_i [\, I \bullet \max_x (M) \bullet \arg(\max_x (M))]}{\sum_i I}$$

c = crisp output value
i = linguistic term
I = inference result
M = membership function of linguistic term

For this example, when the crisp values are fuzzified, the Duty Cycle variable is defined to be mostly "medium" (degree of membership of 0.7) and somewhat "medium fast" (degree of membership 0.1). The arguments for the "medium" and "medium fast" term membership maxima are 165 and 178, respectively.

$$\frac{((0.7 \bullet 1.0 \bullet 165) + (0.1 \bullet 1.0 \bullet 178))}{(0.7 + 0.1)} = 166$$

FIGURE 6: DEFUZZIFICATION OF DUTY CYCLE



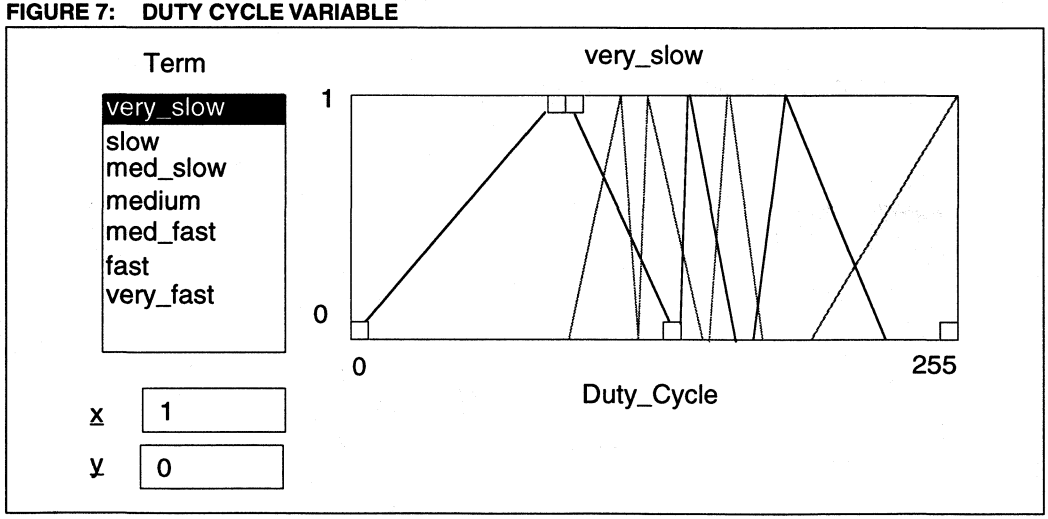
DEBUGGING

In serial debug mode, one can graphically adjust the variable terms and see the results in "real time." On this project, the first variable adjusted was the Duty Cycle variable. Duty Cycle was adjusted so that the beach ball reached 30 inches (Figure 7). The Delta Height terms were fine-tuned -- negative small, zero, and positive small were bunched together -- and the beach ball stabilized at 30 inches (Figure 8). There was virtually no fluctuation in the height. In order for the system to self-correct for environmental (external) changes, the Velocity variable was used. The velocity variable is calculated by the difference in height between consecutive height calculations. A few rules were added that used the Velocity variable to nudge the ball into place when the environmental conditions changed (Figure 9).

Another advantage of fuzzy logic is that it simplifies dealing with non-linearities of the system. The system was highly non-linear, so it was tested at the extremes and moving the beach ball at different rates from one extreme to the other. The Current Height variable needed almost no adjustment (Figure 10). The variable that required the most work was the Duty Cycle variable, but in less than a day, the algorithm was working well within specifications. The beach ball could go from a resting position, with the DC fan off, to the maximum allowable height of 42 inches in less than 8 seconds with no overshoot. Operation between the minimum and maximum height was much quicker, also with no overshoot.

The final graphical representation of the linguistic variables are shown in Figure 7 through Figure 10.

FIGURE 7: DUTY CYCLE VARIABLE



© 1997 Microchip Technology Inc.

FIGURE 8: DELTA HEIGHT VARIABLE

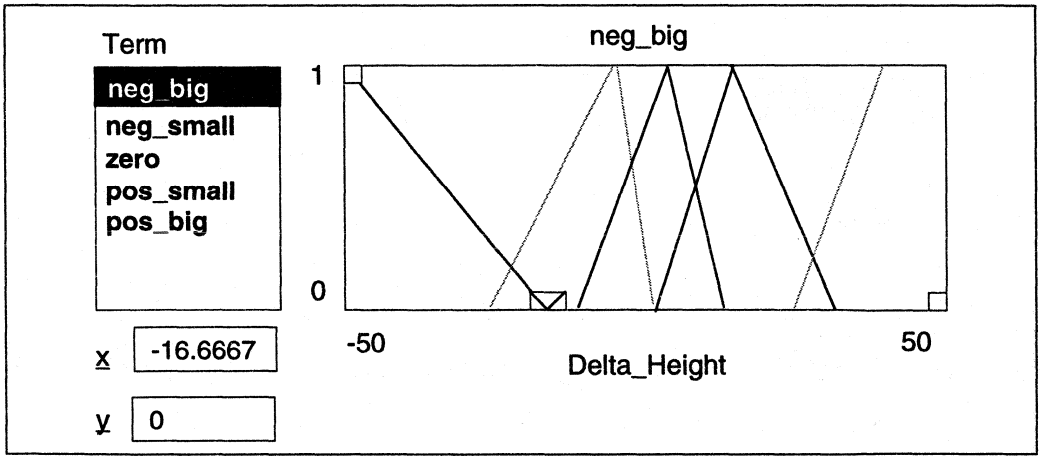
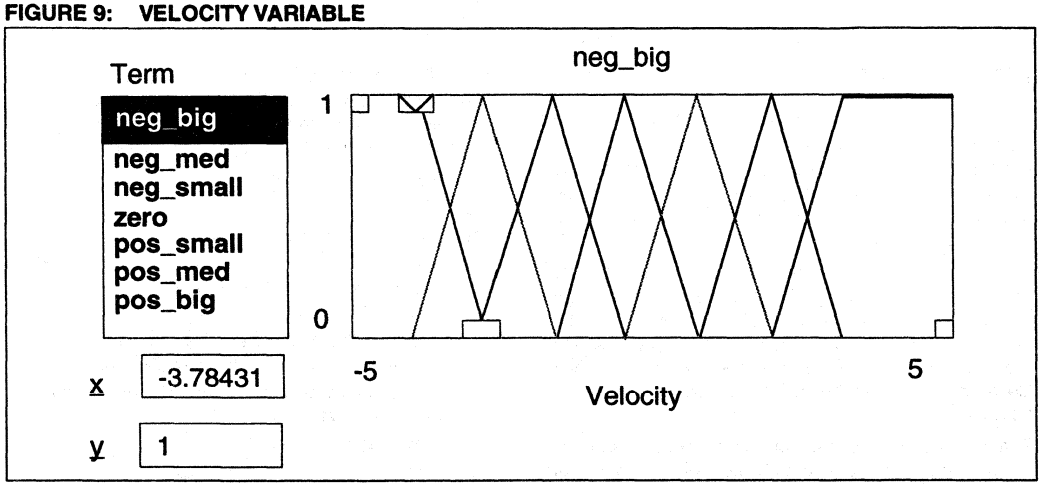
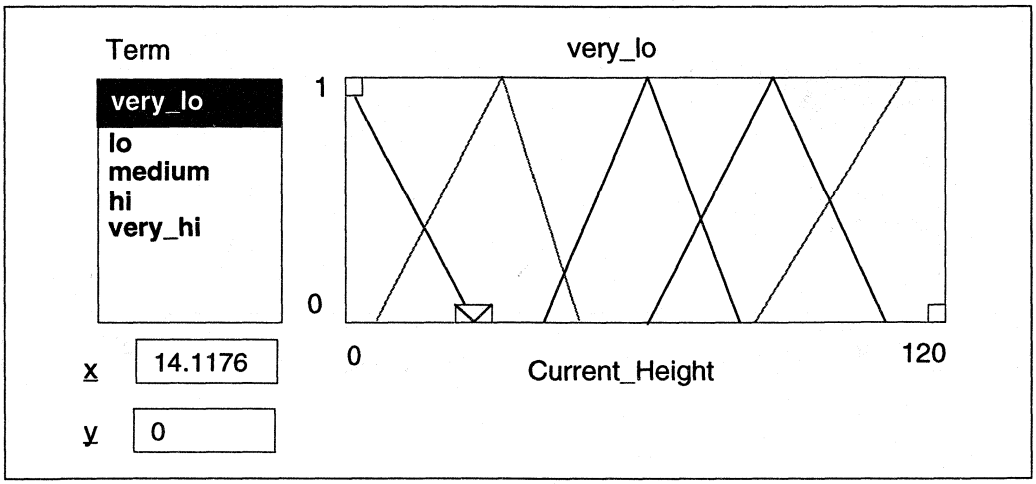


FIGURE 9: VELOCITY VARIABLE



© 1997 Microchip Technology Inc.

FIGURE 10: CURRENT HEIGHT VARIABLE



INTEGRATION

The system parameters and graphical variable representations are captured in a Fuzzy Technology Language (FTL) file. The FTL file is a vendor and hardware independent language which defines the fuzzy logic based system. The FTL file for this project can be seen in Appendix A.

The FTL file is used to generate the public variable definitions and code which can be embedded in the microcontroller. The appropriate device family from the pre-assembler code are generated by simply selecting the compile pull-down menu. Once the pre-assembler file is generated, the "hooks" to the main program must be added.

The best way to embed the code is to use the template `MYMAIN.ASM`. The template for each of the families of devices (PIC16C5X, PIC16CXXX and PIC17CXX) is included in the *fuzzyTECH®-MP* development kit. The template shown in Appendix B is for the PIC16CXXX family.

The file `MYMAIN.ASM` should contain your program in the "`main_loop`" section. The only other modifications required to the template are listed below and are specified in the left hand column of Appendix B.

1. Processor Type definition
2. Code Start Address
3. Fuzzy RAM Start Address
4. Include Public Variable Definition file (myproj.var), which was created by *fuzzyTECH®-MP*
5. Include Pre-Assembler Code (`myproj.asm`) which was created by *fuzzyTECH®-MP*
6. Call Initialization (`initmyproj`) which was created by *fuzzyTECH®-MP*
7. Set Crisp Input Value(s)
8. Call Fuzzy Logic System (`myproj`)
9. Read Crisp Output Value(s)

For this project, the fuzzy logic algorithm assembled to 704 words of program memory and 41 bytes of data memory.

SUMMARY

This project demonstrates many aspects of fuzzy logic control - quick development cycle, real-time debug, sensor integration, and non-linear system control. The total development time for the application took less than a week and performed well within system specifications.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: FUZZY TECHNOLOGY LANGUAGE FILE

```
PROJECT {
  NAME = B_BALL.FTL;
  AUTHOR = ROBERT SCHREIBER;
  DATEFORMAT = M.D.YY;
  LASTCHANGE = 9.16.94;
  CREATED    = 9.14.94;
  SHELL = MP;
  COMMENT {
}  /* COMMENT */
  SHELLOPTIONS {
    ONLINE_REFRESHTIME = 55;
    ONLINE_TIMEOUTCOUNT = 0;
    ONLINE_CODE = OFF;
    TRACE_BUFFER = (OFF, PAR(10000));
    BSUM_AGGREGATION = OFF;
    PUBLIC_IO = ON;
    FAST_CMBF = ON;
    FAST_COA  = OFF;
    SCALE_MBF = OFF;
    FILE_CODE = OFF;
    BTYPE = 8_BIT;
  } /* SHELLOPTIONS */
  MODEL {
    VARIABLE_SECTION {
      LVAR {
        NAME    = current_height;
        BASEVAR = Current_Height;
        LVRANGE = MIN(0.000000), MAX(120.000000),
                  MINDEF(0), MAXDEF(255),
                  DEFAULT_OUTPUT(120.000000);
        RESOLUTION = XGRID(0.000000), YGRID(1.000000),
                     SHOWGRID (ON), SNAPTOGRID(ON);
        TERM {
          TERMNAME = very_lo;
          POINTS = (0.000000, 1.000000),
                   (14.117647, 0.000000),
                   (120.000000, 0.000000);
          SHAPE = LINEAR;
          COLOR = RED (255), GREEN (0), BLUE (0);
        }
        TERM {
          TERMNAME = lo;
          POINTS = (0.000000, 0.000000),
                   (5.176471, 0.000000),
                   (24.941176, 1.000000),
                   (40.941176, 0.000000),
                   (120.000000, 0.000000);
          SHAPE = LINEAR;
          COLOR = RED (0), GREEN (255), BLUE (0);
        }
        TERM {
          TERMNAME = medium;
          POINTS = (0.000000, 0.000000),
                   (27.294118, 0.000000),
                   (51.294118, 1.000000),
                   (66.352941, 0.000000),
                   (120.000000, 0.000000);
          SHAPE = LINEAR;
          COLOR = RED (0), GREEN (0), BLUE (255);
        }
```


© 1997 Microchip Technology Inc.

```
    TERM {
      TERMNAME = hi;
      POINTS = (0.000000, 0.000000),
               (55.529412, 0.000000),
               (82.352941, 1.000000),
               (106.352941, 0.000000),
               (120.000000, 0.000000);
      SHAPE = LINEAR;
      COLOR = RED (128), GREEN (0), BLUE (0);
    }
    TERM {
      TERMNAME = very_hi;
      POINTS = (0.000000, 0.000000),
               (73.411765, 0.000000),
               (113.411765, 1.000000),
               (120.000000, 1.000000);
      SHAPE = LINEAR;
      COLOR = RED (0), GREEN (128), BLUE (0);
    }
  }  /* LVAR */
  LVAR {
    NAME    = delta_height;
    BASEVAR = Delta_Height;
    LVRANGE = MIN(-50.000000), MAX(50.000000),
              MINDEF(0), MAXDEF(255),
              DEFAULT_OUTPUT(-50.000000);
    RESOLUTION = XGRID(0.000000), YGRID(1.000000),
                 SHOWGRID (ON), SNAPTOGRID(ON);
    TERM {
      TERMNAME = neg_big;
      POINTS = (-50.000000, 1.000000),
               (-16.666667, 0.000000),
               (50.000000, 0.000000);
      SHAPE = LINEAR;
      COLOR = RED (255), GREEN (0), BLUE (0);
    }
    TERM {
      TERMNAME = neg_small;
      POINTS = (-50.000000, 0.000000),
               (-21.764706, 0.000000),
               (-6.470588, 1.000000),
               (-0.588235, 0.000000),
               (50.000000, 0.000000);
      SHAPE = LINEAR;
      COLOR = RED (0), GREEN (255), BLUE (0);
    }
    TERM {
      TERMNAME = zero;
      POINTS = (-50.000000, 0.000000),
               (-12.352941, 0.000000),
               (0.196078, 1.000000),
               (13.529412, 0.000000),
               (50.000000, 0.000000);
      SHAPE = LINEAR;
      COLOR = RED (0), GREEN (0), BLUE (255);
    }
    TERM {
      TERMNAME = pos_small;
      POINTS = (-50.000000, 0.000000),
               (0.196078, 0.000000),
               (10.000000, 1.000000),
               (10.392157, 1.000000),
               (32.745098, 0.000000),
               (50.000000, 0.000000);
      SHAPE = LINEAR;
      COLOR = RED (128), GREEN (0), BLUE (0);
```


© 1997 Microchip Technology Inc.

```
  }
  TERM {
    TERMNAME = pos_big;
    POINTS = (-50.000000, 0.000000),
             (26.470588, 0.000000),
             (39.803922, 1.000000),
             (50.000000, 1.000000);
    SHAPE = LINEAR;
    COLOR = RED (0), GREEN (128), BLUE (0);
  }
} /* LVAR */
LVAR {
  NAME     = duty_cycle;
  BASEVAR = Duty_Cycle;
  LVRANGE = MIN(0.000000), MAX(255.000000),
            MINDEF(0), MAXDEF(255),
            DEFAULT_OUTPUT(0.000000);
  RESOLUTION = XGRID(0.000000), YGRID(1.000000),
               SHOWGRID (ON), SNAPTOGRID(ON);
  TERM {
    TERMNAME = very_slow;
    POINTS = (0.000000, 0.000000),
             (1.000000, 0.000000),
             (103.000000, 1.000000),
             (113.000000, 1.000000),
             (147.000000, 0.000000),
             (255.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (255), GREEN (0), BLUE (0);
  }
  TERM {
    TERMNAME = slow;
    POINTS = (0.000000, 0.000000),
             (108.000000, 0.000000),
             (127.000000, 1.000000),
             (131.000000, 0.000000),
             (255.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (0), GREEN (255), BLUE (0);
  }
  TERM {
    TERMNAME = med_slow;
    POINTS = (0.000000, 0.000000),
             (133.000000, 0.000000),
             (142.000000, 1.000000),
             (162.000000, 0.000000),
             (255.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (0), GREEN (128), BLUE (128);
  }
  TERM {
    TERMNAME = medium;
    POINTS = (0.000000, 0.000000),
             (151.000000, 0.000000),
             (164.000000, 1.000000),
             (166.000000, 1.000000),
             (174.000000, 0.000000),
             (255.000000, 0.000000);
    SHAPE = LINEAR;
    COLOR = RED (0), GREEN (0), BLUE (255);
  }
  TERM {
    TERMNAME = med_fast;
    POINTS = (0.000000, 0.000000),
             (166.000000, 0.000000),
             (178.000000, 1.000000),
```


© 1997 Microchip Technology Inc.

```
                (193.000000, 0.000000),
                (255.000000, 0.000000);
      SHAPE = LINEAR;
      COLOR = RED (255), GREEN (0), BLUE (128);
    }
    TERM {
      TERMNAME = fast;
      POINTS = (0.000000, 0.000000),
                (189.000000, 0.000000),
                (202.000000, 1.000000),
                (232.000000, 0.000000),
                (255.000000, 0.000000);
      SHAPE = LINEAR;
      COLOR = RED (128), GREEN (0), BLUE (0);
    }
    TERM {
      TERMNAME = very_fast;
      POINTS = (0.000000, 0.000000),
                (206.000000, 0.000000),
                (255.000000, 1.000000);
      SHAPE = LINEAR;
      COLOR = RED (0), GREEN (128), BLUE (0);
    }
  }  /* LVAR */
  LVAR {
    NAME    = velocity;
    BASEVAR = Velocity;
    LVRANGE = MIN(-5.000000), MAX(5.000000),
              MINDEF(0), MAXDEF(255),
              DEFAULT_OUTPUT(0.000000);
    RESOLUTION = XGRID(0.000000), YGRID(1.000000),
                 SHOWGRID (OFF), SNAPTOGRID(ON);
    TERM {
      TERMNAME = neg_big;
      POINTS = (-5.000000, 1.000000),
                (-3.784314, 1.000000),
                (-2.529412, 0.000000),
                (5.000000, 0.000000);
      SHAPE = LINEAR;
      COLOR = RED (255), GREEN (0), BLUE (0);
    }
    TERM {
      TERMNAME = neg_med;
      POINTS = (-5.000000, 0.000000),
                (-3.784314, 0.000000),
                (-2.529412, 1.000000),
                (-1.274510, 0.000000),
                (5.000000, 0.000000);
      SHAPE = LINEAR;
      COLOR = RED (0), GREEN (255), BLUE (0);
    }
    TERM {
      TERMNAME = neg_small;
      POINTS = (-5.000000, 0.000000),
                (-2.568627, 0.000000),
                (-1.313725, 1.000000),
                (-0.058824, 0.000000),
                (5.000000, 0.000000);
      SHAPE = LINEAR;
      COLOR = RED (0), GREEN (0), BLUE (255);
    }
    TERM {
      TERMNAME = zero;
      POINTS = (-5.000000, 0.000000),
                (-1.000000, 0.000000),
                (-0.019608, 1.000000),
```


© 1997 Microchip Technology Inc.

```
                (0.960784, 0.000000),
                (5.000000, 0.000000);
        SHAPE = LINEAR;
        COLOR = RED (128), GREEN (0), BLUE (0);
      }
      TERM {
        TERMNAME = pos_small;
        POINTS = (-5.000000, 0.000000),
                 (-0.137255, 0.000000),
                 (1.117647, 1.000000),
                 (2.372549, 0.000000),
                 (5.000000, 0.000000);
        SHAPE = LINEAR;
        COLOR = RED (0), GREEN (128), BLUE (0);
      }
      TERM {
        TERMNAME = pos_med;
        POINTS = (-5.000000, 0.000000),
                 (1.078431, 0.000000),
                 (2.333333, 1.000000),
                 (3.588235, 0.000000),
                 (5.000000, 0.000000);
        SHAPE = LINEAR;
        COLOR = RED (0), GREEN (0), BLUE (128);
      }
      TERM {
        TERMNAME = pos_big;
        POINTS = (-5.000000, 0.000000),
                 (2.294118, 0.000000),
                 (3.549020, 1.000000),
                 (5.000000, 1.000000);
        SHAPE = LINEAR;
        COLOR = RED (255), GREEN (0), BLUE (128);
      }
    }  /* LVAR */
  }  /* VARIABLE_SECTION */

  OBJECT_SECTION {
    INTERFACE {
      INPUT = (current_height, FCMBF);
      POS = -213, -137;
      RANGECHECK = ON;
    }
    INTERFACE {
      INPUT = (delta_height, FCMBF);
      POS = -216, -83;
      RANGECHECK = ON;
    }
    INTERFACE {
      OUTPUT = (duty_cycle, COM);
      POS = 158, -79;
      RANGECHECK = ON;
    }
    RULEBLOCK {
      INPUT = current_height, delta_height, velocity;
      OUTPUT = duty_cycle;
      AGGREGATION = (MIN_MAX, PAR (0.000000));
      COMPOSITION = (GAMMA, PAR (0.000000));
      POS = -39, -113;
      RULES {
        IF    current_height = very_lo
          AND delta_height = neg_big
        THEN  duty_cycle = slow   WITH 1.000;
        IF    current_height = very_lo
          AND delta_height = neg_small
        THEN  duty_cycle = med_slow   WITH 1.000;
```

```
IF    current_height = very_lo
  AND delta_height = zero
THEN  duty_cycle = medium   WITH 1.000;
IF    current_height = very_lo
  AND delta_height = pos_small
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = very_lo
  AND delta_height = pos_big
THEN  duty_cycle = very_fast   WITH 1.000;
IF    current_height = lo
  AND delta_height = neg_big
THEN  duty_cycle = slow   WITH 1.000;
IF    current_height = lo
  AND delta_height = neg_small
THEN  duty_cycle = med_slow   WITH 1.000;
IF    current_height = lo
  AND delta_height = zero
THEN  duty_cycle = medium   WITH 1.000;
IF    current_height = lo
  AND delta_height = pos_small
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = lo
  AND delta_height = pos_big
THEN  duty_cycle = very_fast   WITH 1.000;
IF    current_height = medium
  AND delta_height = neg_big
THEN  duty_cycle = very_slow   WITH 1.000;
IF    current_height = medium
  AND delta_height = neg_small
THEN  duty_cycle = med_slow   WITH 1.000;
IF    current_height = medium
  AND delta_height = zero
THEN  duty_cycle = med_fast   WITH 1.000;
IF    current_height = medium
  AND delta_height = pos_small
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = medium
  AND delta_height = pos_big
THEN  duty_cycle = very_fast   WITH 1.000;
IF    current_height = hi
  AND delta_height = neg_big
THEN  duty_cycle = very_slow   WITH 1.000;
IF    current_height = hi
  AND delta_height = neg_small
THEN  duty_cycle = med_slow   WITH 1.000;
IF    current_height = hi
  AND delta_height = zero
THEN  duty_cycle = med_fast   WITH 1.000;
IF    current_height = hi
  AND delta_height = pos_small
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = hi
  AND delta_height = pos_big
THEN  duty_cycle = very_fast   WITH 1.000;
IF    current_height = very_hi
  AND delta_height = neg_big
THEN  duty_cycle = very_slow   WITH 1.000;
IF    current_height = very_hi
  AND delta_height = neg_small
THEN  duty_cycle = slow   WITH 1.000;
IF    current_height = very_hi
  AND delta_height = zero
THEN  duty_cycle = med_slow   WITH 1.000;
IF    current_height = very_hi
  AND delta_height = pos_small
THEN  duty_cycle = medium   WITH 1.000;
```


© 1997 Microchip Technology Inc.

```
IF    current_height = very_hi
  AND delta_height = pos_big
THEN  duty_cycle = very_fast   WITH 1.000;
IF    current_height = very_lo
  AND delta_height = neg_small
  AND velocity = zero
THEN  duty_cycle = very_slow   WITH 1.000;
IF    current_height = very_lo
  AND delta_height = neg_small
  AND velocity = pos_small
THEN  duty_cycle = very_slow   WITH 1.000;
IF    current_height = very_lo
  AND delta_height = neg_small
  AND velocity = pos_med
THEN  duty_cycle = very_slow   WITH 1.000;
IF    current_height = very_lo
  AND delta_height = neg_small
  AND velocity = pos_big
THEN  duty_cycle = very_slow   WITH 1.000;
IF    current_height = very_lo
  AND delta_height = pos_small
  AND velocity = zero
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = very_lo
  AND delta_height = pos_small
  AND velocity = neg_small
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = very_lo
  AND delta_height = pos_small
  AND velocity = neg_med
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = very_lo
  AND delta_height = pos_small
  AND velocity = neg_big
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = lo
  AND delta_height = neg_small
  AND velocity = zero
THEN  duty_cycle = very_slow   WITH 1.000;
IF    current_height = lo
  AND delta_height = neg_small
  AND velocity = pos_small
THEN  duty_cycle = very_slow   WITH 1.000;
IF    current_height = lo
  AND delta_height = neg_small
  AND velocity = pos_med
THEN  duty_cycle = very_slow   WITH 1.000;
IF    current_height = lo
  AND delta_height = neg_small
  AND velocity = pos_big
THEN  duty_cycle = very_slow   WITH 1.000;
IF    current_height = lo
  AND delta_height = pos_small
  AND velocity = zero
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = lo
  AND delta_height = pos_small
  AND velocity = neg_small
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = lo
  AND delta_height = pos_small
  AND velocity = neg_med
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = lo
  AND delta_height = pos_small
  AND velocity = neg_big
```


© 1997 Microchip Technology Inc.

```
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = medium
  AND delta_height = neg_small
  AND velocity = zero
THEN  duty_cycle = slow   WITH 1.000;
IF    current_height = medium
  AND delta_height = neg_small
  AND velocity = pos_small
THEN  duty_cycle = slow   WITH 1.000;
IF    current_height = medium
  AND delta_height = neg_small
  AND velocity = pos_med
THEN  duty_cycle = slow   WITH 1.000;
IF    current_height = medium
  AND delta_height = neg_small
  AND velocity = pos_big
THEN  duty_cycle = slow   WITH 1.000;
IF    current_height = medium
  AND delta_height = pos_small
  AND velocity = zero
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = medium
  AND delta_height = pos_small
  AND velocity = neg_small
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = medium
  AND delta_height = pos_small
  AND velocity = neg_med
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = medium
  AND delta_height = pos_small
  AND velocity = neg_big
THEN  duty_cycle = fast   WITH 1.000;
IF    current_height = hi
  AND delta_height = neg_small
  AND velocity = zero
THEN  duty_cycle = med_slow   WITH 1.000;
IF    current_height = hi
  AND delta_height = neg_small
  AND velocity = pos_small
THEN  duty_cycle = med_slow   WITH 1.000;
IF    current_height = hi
  AND delta_height = neg_small
  AND velocity = pos_med
THEN  duty_cycle = med_slow   WITH 1.000;
IF    current_height = hi
  AND delta_height = neg_small
  AND velocity = pos_big
THEN  duty_cycle = med_slow   WITH 1.000;
IF    current_height = hi
  AND delta_height = pos_small
  AND velocity = zero
THEN  duty_cycle = very_fast   WITH 1.000;
IF    current_height = hi
  AND delta_height = pos_small
  AND velocity = neg_small
THEN  duty_cycle = very_fast   WITH 1.000;
IF    current_height = hi
  AND delta_height = pos_small
  AND velocity = neg_med
THEN  duty_cycle = very_fast   WITH 1.000;
IF    current_height = hi
  AND delta_height = pos_small
  AND velocity = neg_big
THEN  duty_cycle = very_fast   WITH 1.000;
IF    current_height = very_hi
```


© 1997 Microchip Technology Inc.

```
          AND delta_height = neg_small
          AND velocity = zero
        THEN  duty_cycle = medium    WITH 1.000;
        IF    current_height = very_hi
          AND delta_height = neg_small
          AND velocity = pos_small
        THEN  duty_cycle = medium    WITH 1.000;
        IF    current_height = very_hi
          AND delta_height = neg_small
          AND velocity = pos_med
        THEN  duty_cycle = medium    WITH 1.000;
        IF    current_height = very_hi
          AND delta_height = neg_small
          AND velocity = pos_big
        THEN  duty_cycle = medium    WITH 1.000;
        IF    current_height = very_hi
          AND delta_height = pos_small
          AND velocity = zero
        THEN  duty_cycle = very_fast    WITH 1.000;
        IF    current_height = very_hi
          AND delta_height = pos_small
          AND velocity = neg_small
        THEN  duty_cycle = very_fast    WITH 1.000;
        IF    current_height = very_hi
          AND delta_height = pos_small
          AND velocity = neg_med
        THEN  duty_cycle = very_fast    WITH 1.000;
        IF    current_height = very_hi
          AND delta_height = pos_small
          AND velocity = neg_big
        THEN  duty_cycle = very_fast    WITH 1.000;
      }  /* RULES */
    }
    INTERFACE {
      INPUT = (velocity, FCMBF);
      POS = -211, -29;
      RANGECHECK = ON;
    }
  }  /* OBJECT_SECTION */
 }  /* MODEL */
}  /* PROJECT */
TERMINAL {
  BAUDRATE     = 9600;
  STOPBITS     = 1;
  PROTOCOL     = NO;
  CONNECTION   = PORT1;
  INPUTBUFFER  = 4096;
  OUTPUTBUFFER = 1024;
}  /* TERMINAL */
```

APPENDIX B: MYMAIN.ASM TEMPLATE FOR THE PIC16CXXX FAMILY

```
1   PROCESSOR 16C71
;-------------------------------------------------------------------------------
;- USER MAIN FILE                                                               -
;-------------------------------------------------------------------------------
2   CODE_START              EQU     0x100   ;code startadr for 16C71
    RESET_ADR               EQU     0x000   ;reset vector
3   FUZZY_RAM_START         EQU     0x00C   ;first free RAM location for 16C71
4       include             "myproj.var"    ;include preassembler variables
        CBLOCK                              ;starts after fuzzy ram locations
                user1                       ;reserve 1 byte (example)
        ENDC
        ORG CODE_START                      ;example start adress for code
mymain
6       call                initmyproj      ;call init once
main_loop
        movlw               000             ;example
7       movwf               lv0_Input_1     ;set 1st crisp input
        movlw               0A0             ;example
7       movwf               lv1_Input_2     ;set 2nd crisp input
8       call                myproj          ;call preassembler code
        movf                invalidflags,W
        btfss               Z               ;test if the project is completely defined
        goto                case_no_fire
case_fire
        ;proj OK
9       movf                lv2_Output,W    ;fetch crisp output
        ;user code
        goto                main_loop
case_no_fire
        ;no rule defined for this input combination
        ;call default_handling_routine
        ;user code
        goto                main_loop
5       INCLUDE "myproj.asm"                ;include preassembler code
;-------------------------------------------------------------------------------
;- RESET VECTOR                                                                 -
;-------------------------------------------------------------------------------
        ORG                 RESET_ADR
        goto                mymain          ;jump to program code
        END                                 ;end for assembler (only here)
```

Note: Refer to the "Integration" section for the number descriptions.

© 1997 Microchip Technology Inc.

MICROCHIP

AN611

Resistance and Capacitance Meter Using a PIC16C622

Author: *Rodger Richey*
Microchip Technology Inc.

INTRODUCTION

The PIC16C62X devices create a new branch in Microchip's PIC16CXXX 8-bit microcontroller family by incorporating two analog comparators and a variable voltage reference on-chip. The comparators feature programmable input multiplexing from device inputs and an internal voltage reference. The internal voltage reference has two ranges, each capable of 16 distinct voltage levels. Typical applications such as appliance controllers or low-power remote sensors can now be implemented using fewer external components thus reducing cost and power consumption. The 18-pin SOIC or 20-pin SSOP packages are ideal for designs having size constraints.

The PIC16C62X family includes some familiar PIC16CXXX features such as:

- 8-bit timer/counter with 8-bit prescaler
- PORTB interrupt on change
- 13 I/O pins
- Program and Data Memory

Device	Program Memory	Data Memory
PIC16C620	512 x 14	80 x 8
PIC16C621	1K x 14	80 x 8
PIC16C622	2K x 14	128 x 8

This family of devices also introduce on-chip brown-out reset circuitry and a filter on the reset input ($\overline{MCLR}$) to the PIC16CXXX mid-range microcontrollers. Brown-out Reset holds the device in reset while V_{DD} is below the Brown-out Reset voltage of 4.0V, ± 0.2V. The reset filter is used to filter out glitches on the $\overline{MCLR}$ pin.

This application note will describe:

- Comparator module
  - operation
  - initialization
  - outputs
- Voltage Reference module
  - operation
  - initialization
  - outputs
- Linear slope integrating Analog to Digital conversion techniques
  - advantages
  - disadvantages
- Overview of the application circuit
- Detailed description of the measurement techniques used in the application circuit

3

© 1997 Microchip Technology Inc.

COMPARATOR MODULE

The comparator module contains two analog comparators with eight modes of operation. The inputs to the comparators are multiplexed with the RA0 through RA3 pins. The on-chip voltage reference can also be selected as an input to the comparators. The Comparator Control Register (CMCON) controls the operation of the comparator and contains the comparator output bits. Figure 1 shows the CMCON register.

FIGURE 1: CMCON REGISTER

R-0	R-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
C2OUT	C1OUT	—	—	CIS	CM2	CM1	CM0
bit7							bit0

R = Readable bit
W = Writable bit
U = Unimplemented bit, read as '0'
- n =Value at POR reset

bit 7: **C2OUT**: Comparator 2 output
1 = C2 VIN+ > C2 VIN–
0 = C2 VIN+ < C2 VIN–

bit 6: **C1OUT**: Comparator 1 output
1 = C1 VIN+ > C1 VIN–
0 = C1 VIN+ < C1 VIN–

bit 5-4: **Unimplemented:** Read as '0'

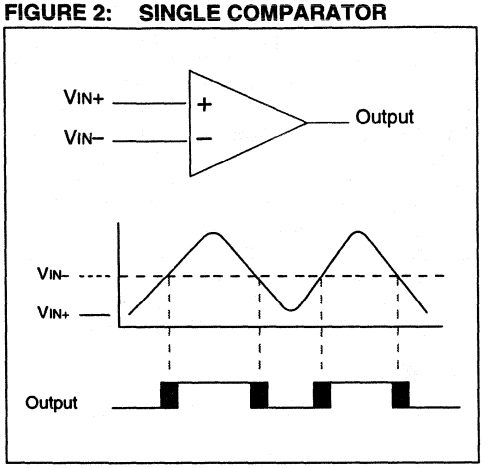
bit 3: **CIS**: Comparator Input Switch
When CM2:CM0: = 001:
1 = C1 VIN– connects to RA3
0 = C1 VIN– connects to RA0
When CM2:CM0 = 010:
1 = C1 VIN– connects to RA3
C2 VIN– connects to RA2
0 = C1 VIN– connects to RA0
C2 VIN– connects to RA1

bit 2-0: **CM2:CM0**: Comparator mode.

© 1997 Microchip Technology Inc.

A single comparator is shown in Figure 2. The relationship between the inputs and the output is also shown. When the voltage at VIN+ is less than the voltage at VIN-, the output of the comparator is at a digital low level. When the voltage at VIN+ is greater than the voltage at VIN-, the output of the comparator is at a digital high level. The shaded areas of the comparator output waveform represent the uncertainty due to input offsets and response time.

FIGURE 2: SINGLE COMPARATOR

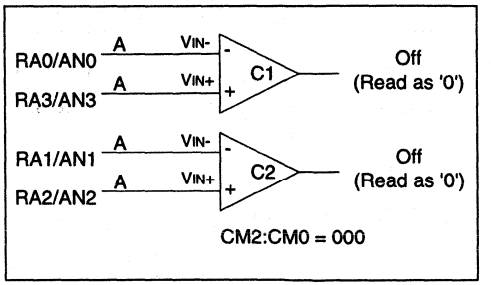


The TRISA register controls the I/O direction of the PORTA pins regardless of the comparator mode. If the comparator mode configures a pin as an analog input and the TRISA register configures that pin as an output, the contents of the PORTA data latch are placed on the pin. The value at the pin, which can be a digital high or low voltage, then becomes the input signal to the comparators. This technique is useful to check the functionality of the application circuit and the comparator module.

Comparator Operating Modes

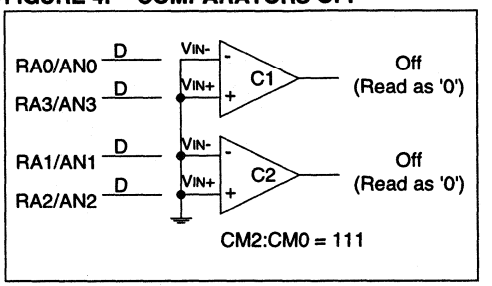
The analog inputs to the comparator module must be between VSS and VDD and one input must be in the Common Mode Range (CMR). The CMR is defined as VDD-1.5 volt to VSS. The output of a comparator will default to a high level if both inputs are outside of the CMR. If the input voltage deviates above VDD or below VSS by more than 0.6 volt, the microcontroller may draw excessive current. A maximum source impedance to the comparators of 10 kΩ is recommended. Figure 3 through Figure 10 show the eight modes of operation.

FIGURE 3: COMPARATORS RESET



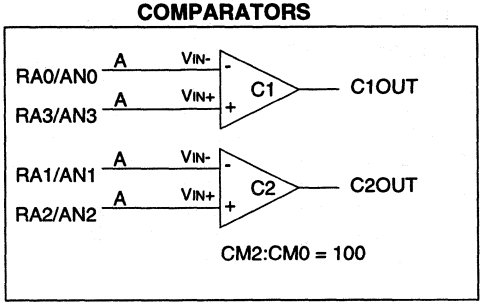
The Comparators Reset Mode (Figure 3) is considered the lowest power mode because the comparators are turned off and RA0 through RA3 are analog inputs. The comparator module defaults to this mode on Power-on Reset.

FIGURE 4: COMPARATORS OFF



The Comparators Off Mode (Figure 4) is the same as the Comparators Reset Mode except that RA0 through RA3 are digital I/O. This mode may consume more current if RA0 through RA3 are configured as inputs and the pins are left floating.

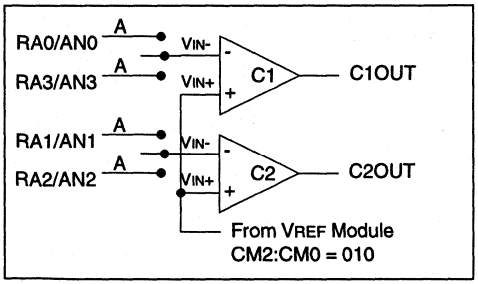
FIGURE 5: TWO INDEPENDENT COMPARATORS



The Two Independent Comparators Mode (Figure 5) enables both comparators to operate independently.

© 1997 Microchip Technology Inc.

FIGURE 6: FOUR INPUTS MULTIPLEXED TO TWO COMPARATORS

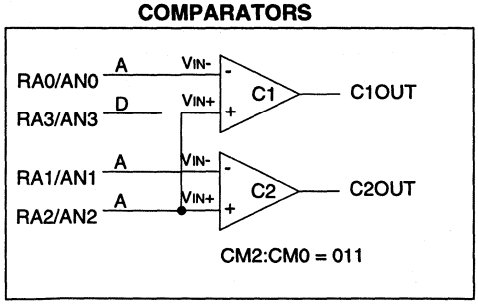


The Four Inputs Multiplexed to Two Comparators Mode (Figure 6) allows two inputs into the VIN- pin of each comparator. The internal voltage reference is connected to the VIN+ pin input of each comparator. The CIS bit, CMCON<3>, controls the input multiplexing to the VIN- pin of each comparator. Table 1 shows this relationship.

TABLE 1: COMPARATOR INPUT MULTIPLEXING

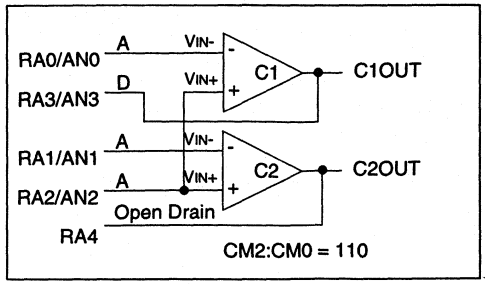
CIS	C1 VIN-	C2 VIN-
0	RA0	RA1
1	RA3	RA2

FIGURE 7: TWO COMMON REFERENCE COMPARATORS



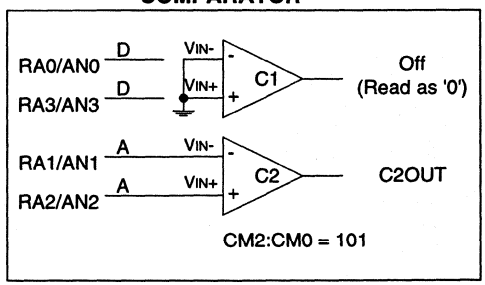
The Two Common Reference Comparators Mode (Figure 7) configures the comparators such that the signal present on RA2 is connected to the VIN+ pin of each comparator. RA3 is configured as a digital I/O pin.

FIGURE 8: TWO COMMON REFERENCE COMPARATORS WITH OUTPUTS



The Two Common Reference Comparators with Outputs Mode (Figure 8) connects the outputs of the comparators to an I/O pin. These outputs are digital outputs only with RA3 defined as a CMOS output and RA4 defined as an open drain output. RA4 requires a pull-up resistor to function properly. The value of resistance used for the pull-up will affect the response time of comparator C2. The signal present on RA2 is connected to the VIN+ pin of both comparators.

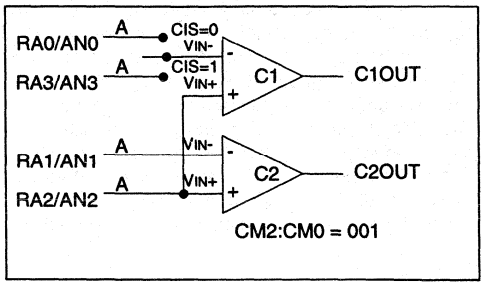
FIGURE 9: ONE INDEPENDENT COMPARATOR



The One Independent Comparator Mode (Figure 9) turns comparator C1 off making both RA0 and RA3 digital I/O. Comparator C2 is operational with analog inputs from RA1 and RA2.

© 1997 Microchip Technology Inc.

FIGURE 10: THREE INPUTS MULTIPLEXED TO TWO COMPARATORS



The Three Inputs Multiplexed to Two Comparators Mode (Figure 10) connects the VIN+ pin of each comparator to RA2. The VIN- pin of comparator 2 is connected to RA1. The CIS bit, CMCON<3>, controls the input to the VIN- pin of comparator 1. If CIS = 0, then RA0 is connected to the VIN- pin. Otherwise RA3 is connected to the VIN- pin of comparator 1.

Note: Each comparator that is active will consume less power when the output is at a high level.

Clearing the Comparator Interrupt Flag

The comparator interrupt flag, CMIF, is located in the PIR1 register. This flag must be cleared after changing comparator modes. Whenever the comparator mode or the CIS bit is changed, the CMIF may be set due to the internal circuitry switching between modes. Therefore, comparator interrupts should be disabled before changing modes. Then, a delay of 10 µs should be used after changing modes to allow the comparator circuitry to stabilize.

The steps to clear the CMIF flag when changing modes are as follows:

- Change the comparator mode or CIS bit
- 10 µs delay
- Read the CMCON register to end the "mismatch" condition
- Clear the CMIF bit of the PIR1 register

The value of C1OUT and C2OUT are internally latched on every read of the CMCON register. The current values of C1OUT and C2OUT are compared with the latched values, and when these values are different a "mismatch" condition occurs. The CMIF interrupt flag will not be cleared if the CMCON register has not been read.

Using the Comparator Module

The CMCON register contains the comparator output bits C1OUT and C2OUT, CMCON<7:6>. These bits are read only. C1OUT and C2OUT follow the output of the comparators and are not synchronized to any internal clock edges. Therefore, the firmware will need to maintain the status of these output bits to determine the actual change that has occurred. The PIR1 register contains the comparator interrupt flag CMIF, PIR1<6>. The CMIF bit is set whenever there is a change in the output value of either comparator relative to the last time the CMCON register was read.

Note: If a change in C1OUT or C2OUT should occur when a read operation on the CMCON register is being executed (start of the Q2 pcycle), the CMIF interrupt flag may not be set.

When reading the PORTA register, all pins configured as analog inputs will read as a '0'. Analog levels on any pin that is defined as a digital input may cause the input buffer to consume more current than is specified.

The code in Example 1 shows the steps required to configure the comparator module. RA3 and RA4 are configured as digital outputs. RA0 and RA1 are configured as the VIN- inputs to the comparators and RA2 is the VIN+ input to both comparators.

EXAMPLE 1: INITIALIZING THE COMPARATOR MODULE

```
CLRF    PORTA           ;init PORTA
MOVLW   0X03            ;Two Common
MOVWF   CMCON           ;Reference
                        ;Comparators
                        ;mode selected
BSF     STATUS,RP0      ;go to Bank 1
MOVLW   0X07            ;Set RA<2:0> as
MOVWF   TRISA           ;inputs,RA<4:3>
                        ;as outputs
BCF     STATUS,RP0      ;go to Bank 0
CALL    DELAY10         ;10µs delay
MOVF    CMCON,F         ;read the CMCON
BCF     PIR1,CMIF       ;clear the CMIF
BSF     STATUS,RP0      ;go to Bank 1
BSF     PIE1,CMIE       ;enable compar-
                        ;ator interrupt
BCF     STATUS,RP0      ;go to Bank 0
BSF     INTCON,PEIE     ;enable global
BSF     INTCON,GIE      ;and peripheral
                        ;interrupts
```

The comparators will remain active if the device is placed in sleep mode, except for the Comparators Off Mode (CM2:CM0 =111) and Comparators Reset Mode (CM2:CM0=000). In these modes the comparators are turned off and are in a low power state. A comparator interrupt, if enabled, will wake-up the device from sleep in all modes except **Off** and **Reset**.

© 1997 Microchip Technology Inc.

Comparator Timings

The comparator module has a response time and a mode change to output valid timing associated with it. The response time is defined as the time from when an input to the comparator changes until the output of that comparator becomes valid. The response time is faster when the output of the comparator transitions from a high level to a low level. The mode change to output valid time refers to the amount of time it takes for the output of the comparators to become valid after the mode has changed. The internal voltage reference may contribute some delay if used in conjunction with the comparators (see Voltage Reference Settling Time).

VOLTAGE REFERENCE MODULE

The voltage reference is a 16-tap resistor ladder network that is segmented to provide two ranges of VREF values. Each range has 16 distinct voltage levels. The voltage reference has a power-down function to conserve power when the reference is not being used. The voltage reference also has the capability to be connected to RA2 as an output. Figure 11 shows the Voltage Reference Control Register (VRCON) register which controls the voltage reference. Figure 12 shows the block diagram for the voltage reference module.

FIGURE 11: VRCON REGISTER

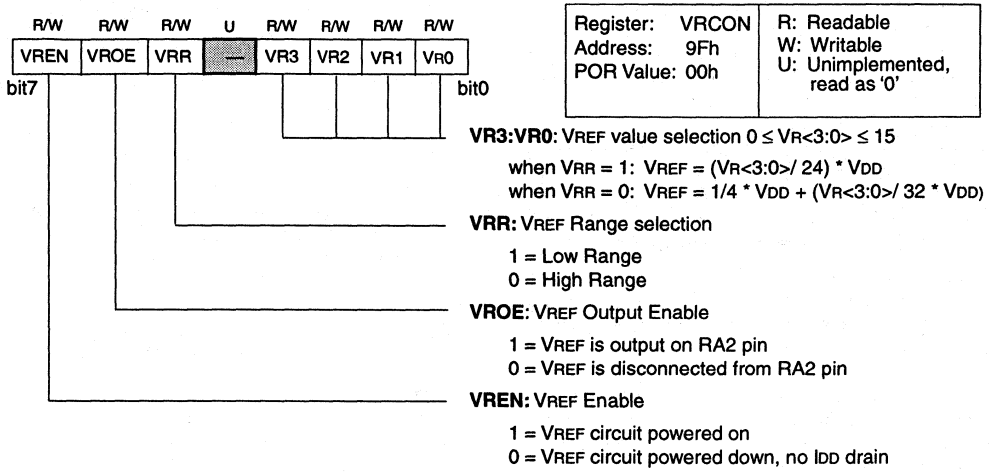
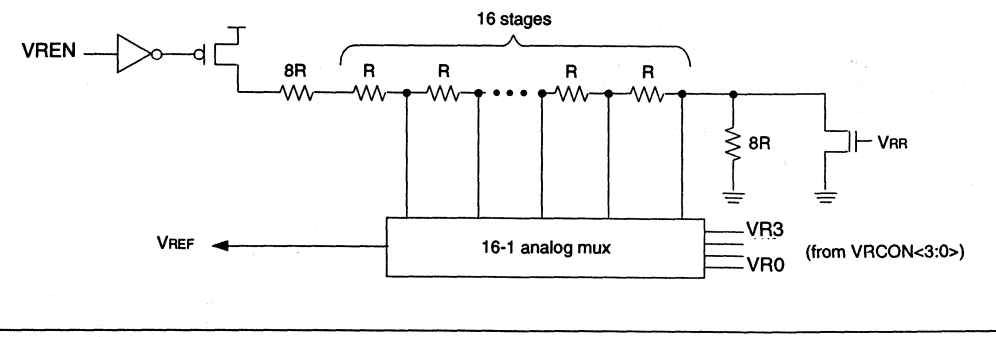


FIGURE 12: VOLTAGE REFERENCE BLOCK DIAGRAM



Note: The voltage reference is VDD derived and therefore, the VREF output changes with fluctuations in VDD.

© 1997 Microchip Technology Inc.

Using the Voltage Reference

The voltage reference module operates independently of the comparator module. The output of the voltage reference may be connected to the RA2 pin at any time by setting the TRISA<2> bit and the VRCON<6> bit (VROE). It should be noted that enabling the voltage reference with an input signal present will increase current consumption. Configuring the RA2 pin as a digital output with the VREF output enabled will also increase current consumption. The increases in current are caused by the voltage reference output conflicting with an input signal or the digital output. The amount of increased current consumption is dependent on the setting of VREF and the value of the input signal or the digital output.

The full range of VSS to VDD cannot be realized due to the construction of the module (Figure 12). The transistors on the top and bottom of the resistor ladder network keep VREF from approaching VSS or VDD. Equation 1 and Equation 2 are used to calculate the output of the voltage reference.

EQUATION 1: VOLTAGE REFERENCE EQUATION, VRR = 1

$$V_{REF} = (VR<3:0>/24) \times V_{DD}$$

EQUATION 2: VOLTAGE REFERENCE EQUATION, VRR = 0

$$V_{REF} = (V_{DD}/4) + (VR<3:0>/32) \times V_{DD}$$

An example of how to configure the voltage reference is given in Equation 2. The reference is set for an output voltage of 1.25V at a VDD of 5.0V.

EXAMPLE 2: VOLTAGE REFERENCE CONFIGURATION

```
MOVLW   0X02        ;4 Inputs Muxed
MOVWF   CMCON       ;to 2 comps.
BSF     STATUS,RP0  ;go to Bank 1
MOVLW   0x07        ;RA3-RA0 are
MOVWF   TRISA       ;outputs
MOVLW   0XA6        ;enable VREF,
MOVWF   VRCON       ;low range
                    ;set VR<3:0>=6
BCF     STATUS,RP0  ;go to Bank 0
CALL    DELAY10     ;10µs delay
```

If the voltage reference is used with the comparator module, the following steps should be followed when making changes to the voltage reference.

1. Disable the comparator interrupts
2. Make changes to the voltage reference
3. Delay 10 µs to allow VREF to stabilize
4. Delay 10 µs to allow comparators to settle
5. Clear the comparator interrupt flag
   - Read the CMCON register
   - Clear the CMIF bit
6. Enable comparator interrupts

The output of the voltage reference may be used as a simple DAC. However, the VREF output has limited drive capability when connected to the RA2 pin. In fact the amount of drive the voltage reference can provide is dependent on the setting of the tap on the resistor ladder. If VREF is used as an output, an external buffer must be used.

Voltage Reference Settling Time

Settling time of the voltage reference is defined as the time it takes the output voltage to settle within 1/4 LSb after making a change to the reference. The changes include adjusting the tap position on the resistor ladder, enabling the output, and enabling the reference itself. If the voltage reference is used with the comparator module, the settling time must be considered.

© 1997 Microchip Technology Inc.

MAKING SIMPLE A/D CONVERSIONS

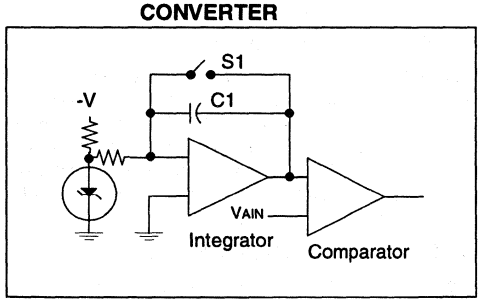
Linear slope integrating A/D converters are very simple to implement and can achieve high linearity and resolution for low conversion rates. The three types of converters that will be discussed are the single-slope, dual-slope, and modified single-slope converters. The following material was referenced from application note AN260, *"A 20-Bit (1ppm) Linear Slope-Integrating A/D Converter"*, found in the Linear Applications Handbook from National Semiconductor®.

Single-Slope Integrating Converter

A single-slope integrating converter is shown in Figure 13. In a single-slope converter, a linear ramp is compared against an unknown input **VAIN**. When the switch **S1** is opened the ramp begins. The time interval between the opening of the switch and the comparator changing state is proportional to the value of **VAIN**.

The basic assumptions are that the integrating capacitor **C1** and the clock used to measure the time interval remain constant over time and temperature. This type of converter is heavily dependent on the stability of the integrating capacitor.

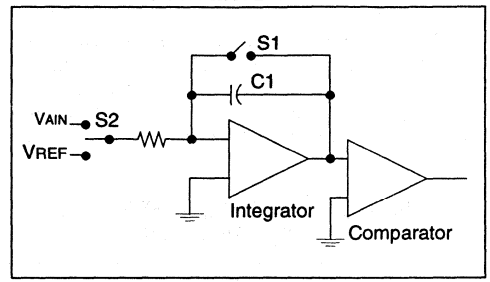
FIGURE 13: SINGLE-SLOPE INTEGRATING CONVERTER



Dual-Slope Integrating Converter

Figure 14 shows a dual-slope integrating converter. The dual-slope converter integrates the VAIN input for a predetermined length of time. The voltage reference is then switched into the integrator input, using S2, which integrates in a negative direction from the VAIN slope. The length of time the reference slope requires to return to zero is proportional to the value of VAIN. Both slopes are made with the same integrating capacitor C1 and measured with the same clock, so they need only to be stable over one conversion cycle.

FIGURE 14: DUAL-SLOPE INTEGRATING CONVERTER



The dual-slope converter essentially removes the stability factor of the integrating capacitor from a conversion, however, the dielectric absorption of C1 has a direct effect. Dielectric absorption not only creates residual non-linearity in the dual-slope converter, but causes the converter to output different values for a fixed input as the conversion rate is varied. Dielectric absorption is defined as the capacitor dielectric's unwillingness to accept or give up charge instantaneously. This effect is modeled as a parasitic RC network across the main capacitor. A charged capacitor will require some time to discharge, even through a dead short, due to the parasitic RC network and some amount of charge will be absorbed by the parasitic C after charging of the main capacitor has stopped. Typically, Teflon, polystyrene and polypropylene dielectrics offer better performance than paper, mylar, or glass. Electrolytics have the worst dielectric absorption characteristics and should be avoided for use in slope integrating converters.

National Semiconductor is a Registered Trademark of National Semiconductor Corporation.

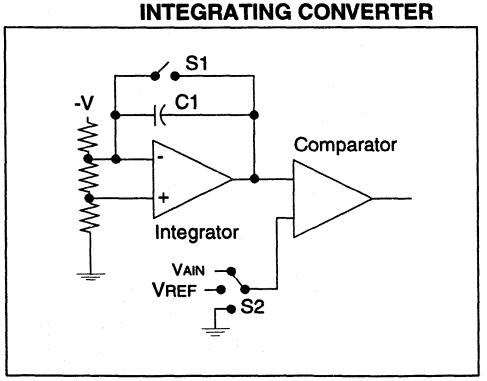
© 1997 Microchip Technology Inc.

Modified Single-Slope Converter

The modified single-slope converter has been designed to compensate for the effects present in the previous converters. Resolutions of up to 16-bits can be achieved using high precision components and voltage reference source. Figure 15 shows the modified single-slope converter. Some features of this converter are:

- Continuously corrects for zero and full-scale drifts in all components of the circuit.
- The integrating capacitor C1 is charged periodically and always in the same direction. The error induced from dielectric absorption will be small and can be compensated by using an offset term in the calibration procedure.
- The ramp voltage always approaches the comparator trip point from the same direction and slew rate.
- There is no noise rejection capability because the input signal is directly coupled to the comparator input. A filter at the comparator input would cause a delay due to the settling time of the filter.

FIGURE 15: MODIFIED SINGLE-SLOPE INTEGRATING CONVERTER



The microcontroller sends a periodic signal to switch S1 regardless of the operating mode of the system. The output of the integrator is a fixed-frequency period and height signal which is fed into the input of the comparator. The time between ramps is long enough to allow integrating capacitor C1 to discharge completely. The other input is multiplexed with ground, reference, and VAIN through switch S2. When the microcontroller starts a conversion, the ground signal is switched into the comparator and the time for the ramp to cross zero is measured and stored. The same measurements are repeated for the reference and VAIN signals. Assuming that the integrator ramps are highly linear, Equation 3 is used to determine the value of VAIN.

EQUATION 3: OUTPUT EQUATION FOR THE MODIFIED-SLOPE CONVERTER

$$V_{AIN} = \frac{t_{AIN} - t_{GND}}{t_{VREF} - t_{GND}} \times K \ mV$$

where τ_{AIN} is the measured time for the VAIN signal, τ_{VREF} is the measured time for the voltage reference signal, τ_{GND} is the measured time for the ground signal, and K is a constant (typically 10^7).

APPLICATION CIRCUIT

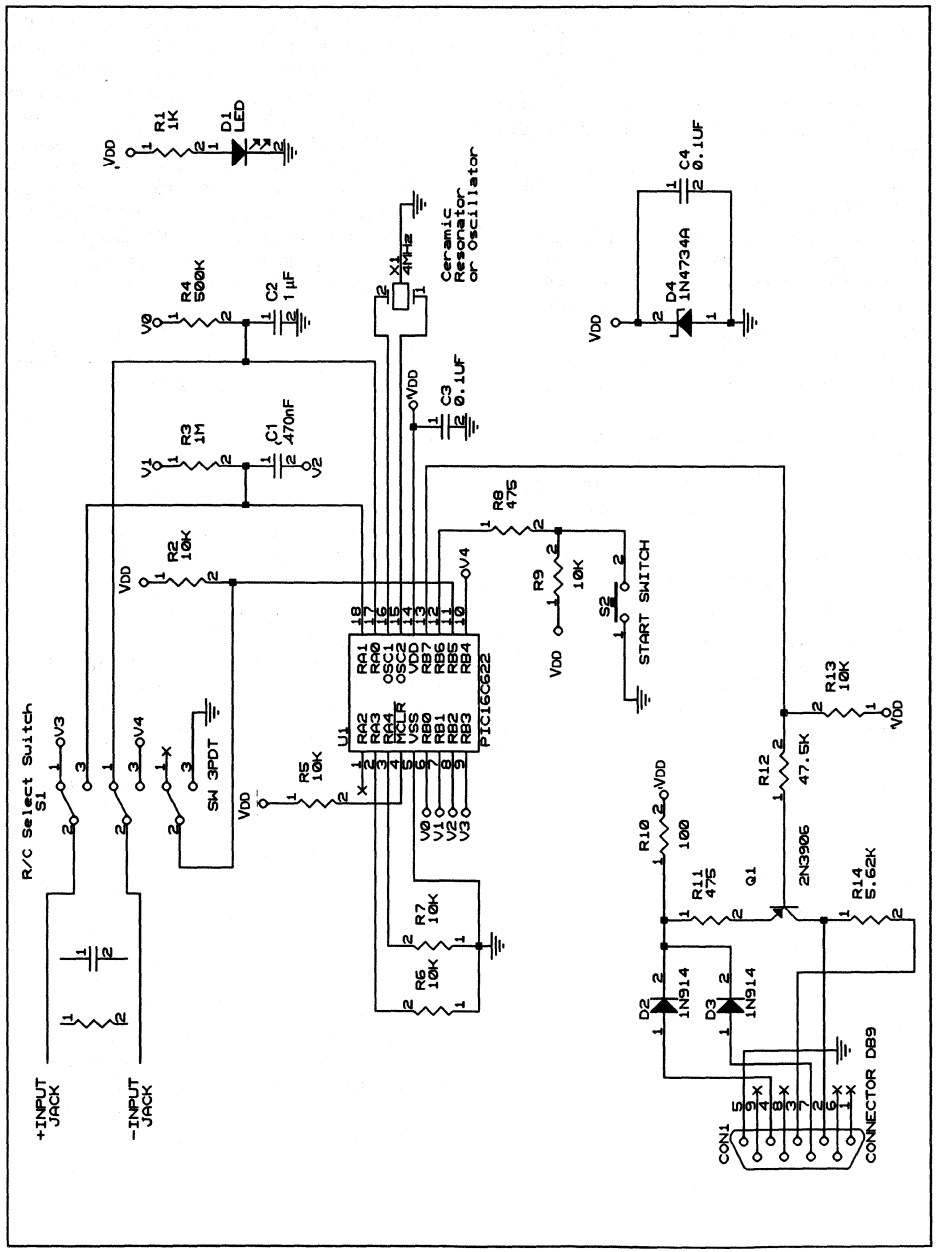
The application circuit, called PICMETER, uses a PIC16C622 as a resistance and capacitance meter. The PICMETER uses a variation of the single-slope integrating converter. The linear slope and integrator of Figure 13 are replaced with the exponential charge waveform of an RC. The charge time of a known component is compared against the charge time of an unknown component to determine the value of the unknown component.

A schematic of the PICMETER is shown in Figure 16. All reference designators cited in this section refer to this schematic. Results are transmitted to a PC which displays the value measured. The PICMETER can measure resistance in the range 1Ω to 999Ω and capacitance from 1 nF to 999 nF.

The following sections describe, in detail, the hardware, firmware, and PC software used in the application circuit. Appendix A shows the PICMETER firmware and Appendix B has the PC software. Appendix C contains the PCB layout.

© 1997 Microchip Technology Inc.

FIGURE 16: PICMETER SCHEMATIC



© 1997 Microchip Technology Inc.

Power

The RS-232 serial port provides power to the PICMETER. The RTS and DTR lines from the serial port output 3V to 11V to the PICMETER. The diodes D2 and D3 prevent any damage to the PC's serial port. Resistor R10 is used to current limit the Zener diode, D4. D4 is used to regulate the RTS and DTR voltage to 5.6V. Capacitors C3 and C4 provide power supply filtering to the Zener diode and the PIC16C622. This method of supplying power to devices using a serial port, such as a trackball or mouse, is very simple considering that the PICMETER requires approximately 7 mA to function.

Switches

Switch S1 is used to select either a resistor or capacitor measurement. RB5 of the PIC16C622 is used to detect what type of component is being measured. This switch also swaps the unknown component into the RC network.

If a resistor is the unknown component and a capacitor measurement is requested, the circuit reduces to a resistor divider on the VIN- pin of the comparator. This would result in a measured value of 0 pF if the voltage on the resistor divider network is greater than the voltage reference setting. Otherwise an error is detected. If a capacitor is the unknown component and a resistor measurement is selected, the circuit reduces to a capacitor divider network on the VIN- pin of the comparator. This case will also produce an error message.

Resistor measurements that are started without any component connected to the measuring terminals will cause an error. Capacitor measurements without a component connected to the measuring terminals will give a result of 0 pF.

Switch S2 is used to initiate a measurement. The switch is connected to RB6 of the PIC16C622 and the PORTB wake-up on change interrupt is used to detect a key press. A modified version of the firmware in AN552, *"Implementing Wake-up on Key Stroke"* was used to control the interrupt.

Measuring the Charge Time

The procedures for measuring a resistor or capacitor are the same except for the I/O pins used to control the RC networks. This also applies when measuring a known or unknown component.

Measurement Overview

The charge time of the unknown RC network is measured using Timer0. This value is multiplied by the known value of resistance or capacitance and stored in an accumulator. Then the charge time of the known RC network is measured. The accumulator is divided by the known RC network charge time to give the value of resistance or capacitance of the unknown component. Equation 4 shows the equation used to calculate resistance and Equation 5 shows the capacitance equation.

EQUATION 4: RESISTANCE EQUATION

$$R_{UNK} = \frac{t_{UNK} \times R_{KN}}{t_{KN}}$$

EQUATION 5: CAPACITANCE EQUATION

$$C_{UNK} = \frac{t_{UNK} \times C_{KN}}{t_{KN}}$$

RUNK and CUNK are the unknown resistor and capacitor values. RKN and CKN are the known resistor and capacitor values. τ_{UNK} and τ_{KN} are the charge times for the unknown and known components.

© 1997 Microchip Technology Inc.

Detailed Measurement Description

The first step in measuring the charge time of either the known or the unknown RC networks is to reconfigure the I/O pins. The default state of the PORTA and PORTB pins connected to the RC network are all grounded outputs. This discharges all capacitors in the RC networks. The unknown component is measured first, so the known component, R4 or C1, is removed from the RC network. This is accomplished by making RB0 or RB2 on the PIC16C622 an input. Connections to the other RC network are kept grounded.

The analog modules are now initialized. The mode of the comparators is set to Four Inputs Multiplexed to Two Comparators (Figure 6). The CIS bit, CMCON<3> is cleared to select RA0 as the VIN- input to comparator 1 and RA1 as the VIN- input to comparator 2. The voltage reference is enabled, the output is disabled, and the high range is selected. The tap on the resistor ladder is set to 12. The value of 12 was selected because it is the lowest value of VREF that will trip the comparators, yet gives a time constant long enough to achieve good resolution for the measurement. After a 20 msec delay, which allows the analog modules to stabilize, the comparator flag is cleared. Comparator interrupts are enabled and Timer0 is cleared. Finally, the PEIE bit is set to enable comparator interrupts and the GIE bit is set to enable interrupts.

Now that the analog systems are ready, Timer0 is cleared again and power is applied to the unknown RC network by setting RB1 or RB3 high. Timer0 begins to increment a set of three registers which are cascaded together. These registers contain the charge time of the component. While waiting for the DONE flag, the ERROR flag is checked. See the Error Message section for an explanation of error detection. When the capacitor voltage trips the comparator, Timer0 is prevented from further incrementing the time registers and the DONE flag is set. The value in the time registers is τ_{UNK}.

The analog modules are now disabled. The comparator interrupts are disabled and the comparators are turned off (CM2:CM0=111). RA0 through RA3 and RB0 through RB4 are set up as grounded outputs to discharge the capacitors in the RC networks. This prevents a false reading during the next measurement. The voltage reference is disabled to conserve power and all interrupt flags are cleared. Extra delay loops are added at this time to ensure that the capacitors are discharged.

The charge time, τ_{UNK}, is then multiplied by the value of known resistance or capacitance. These values, in pF or Ω, were obtained by measuring the known RC networks with a Fluke meter. Each of these values is a 24-bit number. The result of multiplication is a 56-bit number which is stored in accumulators ACCb (most significant 24-bits) and ACCc (least significant 24-bits).

The process now repeats itself, except this time the charge time of the known RC network is measured. Now the unknown component is removed from the RC network by making the connections from the PIC16C622 inputs. The analog modules are initialized and the same procedure explained above is followed to measure the charge time of the known RC network. The 56-bit result previously stored in accumulators ACCb and ACCc is now divided by the charge time of the known component, τ_{KN}. This result is a 24-bit number which has the units of pF or Ω. This value is then transmitted to the PC.

© 1997 Microchip Technology Inc.

RS-232 Transmission

PICMETER uses a transmit only, software implemented serial port adapted from AN593, *"Serial Port Routines Without Using the TMR0"*. Hardware hand-shaking is not used. Since the serial port is realized in software, all interrupts must be disabled during transmission otherwise the baud rate can get corrupted.

On power-up, PICMETER sends a boot message to the PC which is "PICMETER Booted!". Otherwise, a four byte packet structure with a command byte and 3 data bytes is used. The command byte contains one of four possible commands:

- ASCII '**S**' signifies that a measurement has been initiated
- ASCII '**E**' tells the PC that an **E**rror has been detected
- ASCII '**R**' tells the PC that **R**esistance data is contained in the three data bytes
- ASCII '**C**' tells the PC that **C**apacitance data is contained in the three data bytes

The first data byte for the '**R**' and '**C**' commands contain the MSB of the measured value. The last data byte contains the LSB of the measured value. The three data bytes for the commands '**S**' and '**E**' do not contain any useful information at this time.

An 'S' command is issued every time the start switch, S2, is pressed. PICMETER then sends an 'R' or 'C' command for a valid measurement or an 'E' command when an error is detected.

Since the PICMETER operates from a single supply voltage, a discrete transistor is used as a level shifter. This insures that a low output on the RS-232 TXD line is between -3V and -11V. When the TXD line, RB7, from the PIC16C622 is at a logic high level, the transistor Q1 is off. The RXD line of the computer will then be at approximately the same voltage as the TXD line, -11V to -3V. A logic low level from RB7 of the PIC16C622 will turn on transistor Q1. This will bring the RXD line of the computer to about the same voltage of the DTR or RTS line, +3V to +11V.

The pins of interest on the DB9 connector CON1 are:

- pin 2 - RXD
- pin 3 - TXD
- pin 4 - DTR
- pin 5 - GND
- pin 7 - RTS

RTS, DTR, and GND provide power and ground to the PICMETER. RXD is connected to the collector of transistor Q1. TXD is connected to RXD through resistor R14. Since hardware hand-shaking is not implemented on the PICMETER, DSR (pin 6) and CTS (pin 8) are left disconnected.

The demo board developed by Microchip was intended to connect directly to a 9-pin serial port. A 9-pin male-to-female cable may also be used. These boards were manufactured by Southwest Circuits located in Tucson, Arizona (Appendix C). The PCB layout for this demo board is shown in Appendix C.

Error Message

The error message is sent only when the PICMETER is making a measurement and detects an error. The range of resistance that the PICMETER measures is 1 kΩ to 999 kΩ. Using the value of C2, 1 µF, the range of charging times for resistance measurements is 1msec to 999 ms. The range of capacitor charging times is also 1 ms to 999 ms using the resistance value of R3, 1MΩ, and a capacitor measuring range of 1 nF to 999 nF. A ceramic resonator of 4 MHz gives Timer0 a resolution of 1 ns. Therefore, the highest count that the time registers should reach is 999,000. This is a 20-bit number. If the 21st bit should ever be set, it is assumed that the PICMETER is trying to measure the air gap between the measuring terminals, a component that is out of range, or switch S1 is not set correctly for the component in the measuring terminals.

24-Bit Math Routines

The 24-bit math routines were developed using simple algorithms found in any computer math book. These math routines include addition, subtraction, multiplication, division, and 2's complement. Four 24-bit accumulators located in the general purpose RAM area of the PIC16C622 are used by the math routines: ACCa, ACCb, ACCc, and ACCd. Table 2 shows the relationship between the math routines and the accumulators.

TABLE 2: MATH ROUTINE ACCUMULATORS

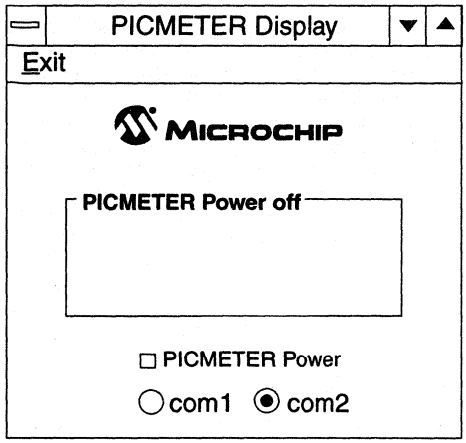
Name	Operation	Result	Temp. Storage
Add	ACCa + ACCb	ACCb	N/A
Subtract	2's Comp ACCa then	ACCa	N/A
	ACCa + ACCb	ACCa	
Multiply	ACCa x ACCb	ACCb (MSB's) ACCc (LSB's)	ACCd
Divide	ACCb:ACCc ACCa	quotient in ACCc remainder in ACCb	ACCd
2's Comp	NOT(ACCa) + 1	ACCa	N/A

© 1997 Microchip Technology Inc.

Computer Program

The program that receives data from the PICMETER was written in Visual Basic® from Microsoft® for the Windows® environment. Figure 17 show the display of the Windows based PICMETER program.

FIGURE 17: PICMETER PC PROGRAM



The operation of this program is simple. A functional description is given below:

a) Select the appropriate COM port by clicking on the COM1 or COM2 buttons.

b) Turn power on to the PICMETER by clicking on the PICMETER Power button.

c) The frame message should read "PICMETER Booted!", the frame contents will be cleared, and the LED on the PICMETER should be on.

d) The switch S1 selects the type of component that is in the measuring terminals.

e) Pressing the START button, S2, on the PICMETER will initiate a measurement. The frame message should read "Measuring Component" and the contents of the frame will be cleared.

f) When the measurement is complete, the frame message will read "Resistance" or "Capacitance" depending on the position of switch S1. The value of the component will be displayed in the frame as well as the units.

g) If an error is detected, the frame message will read "Error Detected". This is only a measurement error. Check the component on the measuring terminals and the position of switch S1.

h) Turn off the PICMETER by clicking on the PICMETER Power button. The frame message will change to "PICMETER Power OFF", the frame contents will be cleared, and the LED on the PICMETER will turn off.

Appendix B contains a complete listing of the Visual Basic program.

© 1997 Microchip Technology Inc.

PICMETER ACCURACY

The PICMETER measures capacitance in the range of 1 nF to 999 nF. Table 3 shows a comparison of various capacitors. All capacitors have a tolerance of 10% and have various dielectrics. The average error percentage is 3%.

TABLE 3: CAPACITANCE MEASUREMENTS

Capacitance Accuracy			
Marked Value	**Fluke Value**	**PICMETER Value**	**Error %**
2.2 nF	2.3 nF	2.2 nF	4.3
2.5 nF	2.63 nF	2.5 nF	4.9
20 nF	16.5 nF	16.3 nF	1.2
33 nF	35.2 nF	35.8 nF	1.7
47 nF	45 nF	44.5 nF	1.1
50 nF	52 nF	52.9 nF	1.7
100 nF	99.7 nF	93 nF	6.7
0.1 µF	95 nF	96.1 nF	1.2
0.1 µF	99.4 nF	102.8 nF	3.4
0.22 µF	215 nF	215.2 nF	0.1
470 nF	508 nF	518.9 nF	2.1
940 nF	922 nF	983.1 nF	6.6

The 2.5 nF, 100 nF and 940 nF capacitors all have polyester dielectric material. The Equivalent Series Resistance (ESR) of polyester capacitors is typically high which would cause the PICMETER to have a larger error than other dielectrics. If the error percentages for these capacitors is ignored, the average error decreases to 1.9%.

The resistance range of the PICMETER is 1 kΩ to 999 kΩ. Table 4, Resistance Measurements, shows a comparison of various resistors in this range. All resistors have a tolerance of 5%. The average error percentage is 1%.

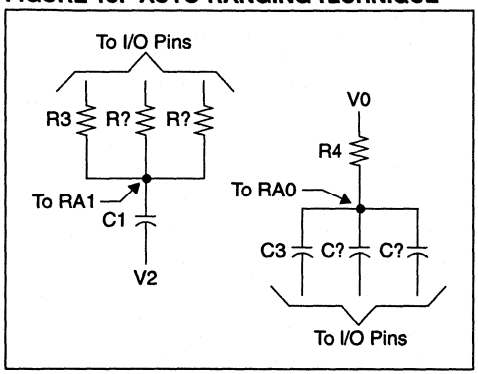
TABLE 4: RESISTANCE MEASUREMENTS

Resistance Accuracy			
Marked Value	**Fluke Value**	**PICMETER Value**	**Error %**
1.2k	1.215k	1.2k	1.3
5.1k	5.05k	5.0k	1.0
8.2k	8.47k	8.3k	2.0
10k	10.2k	10k	2.0
15k	15.36k	15.1k	1.7
20k	20.8k	20.5k	1.5
30k	30.4k	30k	1.4
51k	50.3k	49.8k	1.0
75k	75.5k	74.8k	1.0
91k	96.4k	95.9k	0.6
150k	146.3k	145.6k	0.5
200k	195.5k	195k	0.3
300k	309k	309.5k	0.2
430k	433k	434.5k	0.4
560k	596k	599.6k	0.6
680k	705k	709.8k	0.7
820k	901k	907.3k	0.7
910k	970k	977.8k	0.8

© 1997 Microchip Technology Inc.

The accuracy of the PICMETER is dependent on the range of components being measured. If auto-ranging could be implemented, the accuracy of the PICMETER could be improved. The addition of capacitors in parallel with C2 of Figure 16 would allow auto-ranging for resistor measurements. Additional resistors in parallel with R3 would give auto-ranging capability to capacitor measurements. Figure 18 shows a simple implementation of auto-ranging given that the I/O pins are available. The R? and C? are the extra components that are added to the PICMETER circuit. These components should be optimized for a particular range of devices.

FIGURE 18: AUTO-RANGING TECHNIQUE



Another addition to the PICMETER that would increase the accuracy of components being measured is a constant current source. The source would feed into the resistor of the RC networks. This provides the same charging current to all RC networks being measured. Figure 19 shows a bilateral current source and Figure 20 shows a precision current source.

FIGURE 19: BILATERAL CURRENT SOURCE

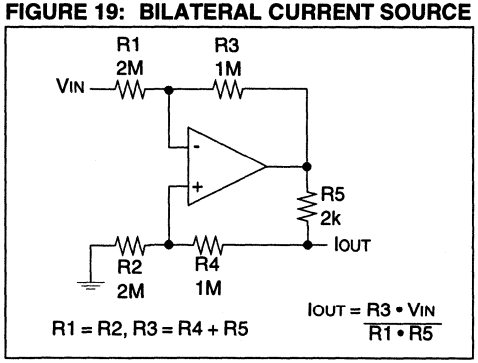
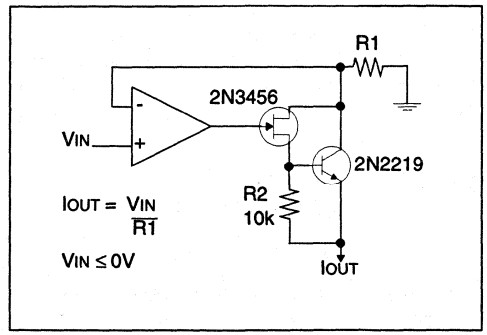
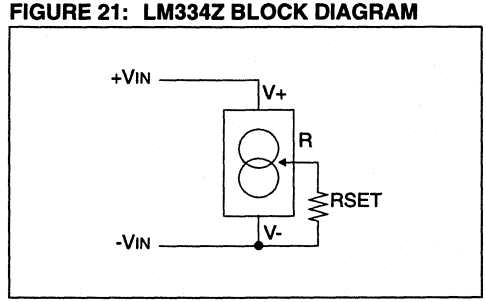


FIGURE 20: PRECISION CURRENT SOURCE



An alternative to the previous current sources is a single chip solution. A 3-terminal adjustable current source, such as a LM134/LM234/LM334 from National Semiconductor, is an ideal choice. This output current is programmable from 1 µA to 10 mA and requires a single external resistor to set the value of current. Figure 21 shows a block diagram of the LM334Z.

FIGURE 21: LM334Z BLOCK DIAGRAM



CONCLUSION

PIC16C62X devices add two significant analog features to the PIC16CXXX mid-range family: comparators and a voltage reference. The flexibility of eight operating modes for the comparator module allows the designer to tailor the PIC16C62X device to the application. The addition of an on-chip voltage reference simplifies the design by removing at least one external component and power consumption. These analog modules coupled with the PIC16CXXX mid-range family core create a new path to achieve high resolution results.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: PICMETER FIRMWARE

```
MPASM 01.40 Released          PICMETER.ASM   1-16-1997  17:49:07         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

             00001         TITLE "PICMETER Firmware for PIC16C622"
             00002         LIST P = 16C622
             00003         ERRORLEVEL  -302
             00004
             00005         INCLUDE   <P16C622.INC>
             00001         LIST
             00002 ; P16C622.INC  Standard Header File, Version 1.01    Microchip Technology, Inc.
             00165         LIST
             00006
2007 3FB1    00007         __CONFIG   _BODEN_OFF & _CP_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC
             00008
             00009 ;*****************************************************************************
             00010 ;*-------------------------------------------------------------------------*
             00011 ;*-                                                                       -*
             00012 ;*-     PICMETER - Resistance and Capacitance Meter                       -*
             00013 ;*-                                                                       -*
             00014 ;*-------------------------------------------------------------------------*
             00015 ;*-                                                                       -*
             00016 ;*-     Author:     Rodger Richey                                         -*
             00017 ;*-                 Applications Engineer                                 -*
             00018 ;*-     Filename:   picmeter.asm                                          -*
             00019 ;*-     Revision:   1 May 1995                                            -*
             00020 ;*-                 1-16-97   ; For compatibility with MPASMWIN 1.40      -*
             00021 ;*-                                                                       -*
             00022 ;*-------------------------------------------------------------------------*
             00023 ;*-                                                                       -*
             00024 ;*- PICMETER is based on a PIC16C622 which has two comparators and a      -*
             00025 ;*- variable voltage reference.  Resistance and capacitance is            -*
             00026 ;*- calculated by measuring the time constant of a RC network.  The      -*
             00027 ;*- toggle switch selects either resistor or capacitor input. The        -*
             00028 ;*- pushbutton switch starts a measurement.  The time constant of the    -*
             00029 ;*- unknown component is compared to that of known component to          -*
             00030 ;*- calculate the value of the unknown component.  The following        -*
             00031 ;*- formulas are used:                                                   -*
             00032 ;*-                                                                       -*
             00033 ;*-     Resistance:     Ru = ( Rk * Tu ) / Tk                           -*
             00034 ;*-     Capacitance:    Cu = ( Ck * Tu ) / Tk                           -*
             00035 ;*-                                                                       -*
             00036 ;*-------------------------------------------------------------------------*
             00037 ;*****************************************************************************
             00038
             00039
             00040 ;*****************************************************************************
             00041 ;*-------------------------------------------------------------------------*
             00042 ;*-     RS232 code borrowed from Application Note AN593                   -*
             00043 ;*-     "Serial Port Routines Without Using Timer0"                     -*
             00044 ;*-     Author: Stan D'Souza                                              -*
             00045 ;*-------------------------------------------------------------------------*
             00046 ;*****************************************************************************
  003D0900   00047 xtal     equ      .4000000
  00002580   00048 baud     equ      .9600
  000F4240   00049 fclk     equ      xtal/4
             00050 ;*****************************************************************************
             00051 ;The value baudconst must be a 8-bit value only
  00000020   00052 baudconst        equ      ((fclk/baud)/3-2)
```

```
                00053 ;***********************************************************************
                00054
                00055
                00056 ;***********************************************************************
                00057 ;       Bit Equates
                00058 ;***********************************************************************
00000000        00059 BEGIN   equ     0               ;begin a measurement flag
00000007        00060 DONE    equ     7               ;done measuring flag
00000005        00061 WHICH   equ     5               ;R or C measurement flag
00000003        00062 F_ERROR equ     3               ;error detection flag
00000005        00063 EMPTY   equ     5               ;flag if component is connected
00000000        00064 V0      equ     0               ;power for R reference ckt
00000001        00065 V1      equ     1               ;power for C reference ckt
00000002        00066 V2      equ     2               ;ground for C reference ckt
00000003        00067 V3      equ     3               ;power for unknown R ckt
00000004        00068 V4      equ     4               ;ground for unknown C ckt
00000007        00069 msb_bit equ     7               ;define for bit 7
00000000        00070 lsb_bit equ     0               ;define for bit 0
00000007        00071 RkHI    equ     0x07            ;value of the known resistance, R4, in ohms
0000009D        00072 RkMID   equ     0x9D            ;measured by a Fluke meter
00000038        00073 RkLO    equ     0x38
00000007        00074 CkHI    equ     0x07            ;value of the known capacitance, C1, in pF
00000047        00075 CkMID   equ     0x47            ;measured by a Fluke meter
00000048        00076 CkLO    equ     0x48
                00077
                00078 ;***********************************************************************
                00079 ;       User Registers
                00080 ;***********************************************************************
                00081 ;       Bank 0
00000020        00082 W_TEMP  equ     0x20            ;Bank 0 temporary storage for W reg
00000021        00083 STATUS_TEMP equ 0x21            ;temporary storage for STATUS reg
00000023        00084 Ttemp   equ     0x23            ;temporary Time register
00000024        00085 flags   equ     0x24            ;flags register
00000025        00086 count   equ     0x25            ;RS232 register
00000026        00087 txreg   equ     0x26            ;RS232 data register
00000027        00088 delay   equ     0x27            ;RS232 delay register
00000028        00089 offset  equ     0x28            ;table position register
00000029        00090 msb     equ     0x29            ;general delay register
0000002A        00091 lsb     equ     0x2A            ;general delay register
00000040        00092 TimeLO  equ     0x40            ;Time registers
00000041        00093 TimeMID equ     0x41
00000042        00094 TimeHI  equ     0x42
                00095
                00096 ;       Math related registers
00000050        00097 ACCaHI  equ     0x50            ;24-Bit accumulator a
00000051        00098 ACCaMID equ     0x51
00000052        00099 ACCaLO  equ     0x52
00000053        00100 ACCbHI  equ     0x53            ;24-Bit accumulator b
00000054        00101 ACCbMID equ     0x54
00000055        00102 ACCbLO  equ     0x55
00000056        00103 ACCcHI  equ     0x56            ;24-Bit accumulator c
00000057        00104 ACCcMID equ     0x57
00000058        00105 ACCcLO  equ     0x58
00000059        00106 ACCdHI  equ     0x59            ;24-Bit accumulator d
0000005A        00107 ACCdMID equ     0x5A
0000005B        00108 ACCdLO  equ     0x5B
0000005C        00109 temp    equ     0x5C            ;temporary storage
                00110
                00111 ;       User Registers Bank 1
                00112 ;W_TEMP  equ     0xA0            ;Bank 1 temporary storage for W reg
                00113
                00114 ;       User defines
                00115 #define tx      PORTB,7         ;define for RS232 TXD output pin
                00116
                00117 ;***********************************************************************
                00118
```


© 1997 Microchip Technology Inc.

```
0000             00119          org     0x0
0000 2810        00120          goto    init
                 00121
0004             00122          org     0x4
0004 28B9        00123          goto    ServiceInterrupts
                 00124
0010             00125          org     0x10
0010             00126 init
0010 1283        00127          bcf     STATUS,RP0      ;select bank 0
0011 0185        00128          clrf    PORTA           ;clear PORTA and PORTB
0012 0186        00129          clrf    PORTB
0013 1786        00130          bsf     tx              ;set TXD output pin
0014 01A4        00131          clrf    flags           ;clear flags register
0015 3010        00132          movlw   0x10            ;load table offset register
0016 00A8        00133          movwf   offset
0017 018B        00134          clrf    INTCON          ;clear interrupt flags and disable interrupts
0018 3007        00135          movlw   0x07            ;turn off comparators, mode 111
0019 009F        00136          movwf   CMCON
001A 2140        00137          call    delay20         ;wait for comparators to settle
001B 089F        00138          movf    CMCON,F
001C 130C        00139          bcf     PIR1,CMIF
001D 1683        00140          bsf     STATUS,RP0      ;select bank 1
001E 3088        00141          movlw   0x88            ;WDT prescaler,internal TMR0 increment
001F 0081        00142          movwf   OPTION_REG
0020 0185        00143          clrf    TRISA           ;PORTA all outputs, discharges RC ckts
0021 3060        00144          movlw   0x60            ;PORTA<7,4:0> outputs, PORTA<6:5> inputs
0022 0086        00145          movwf   TRISB
0023 300C        00146          movlw   0x0C            ;setup Voltage Reference
0024 009F        00147          movwf   VRCON
0025 1283        00148          bcf     STATUS,RP0      ;select bank 0
0026 3008        00149          movlw   0x08            ;enable RBIE interrupt
0027 008B        00150          movwf   INTCON
0028 213D        00151          call    vlong           ;delay before transmitting boot message
0029 213D        00152          call    vlong           ;to allow computer program to setup
002A 213D        00153          call    vlong
002B 2131        00154          call    BootMSG         ;transmit boot message
002C 178B        00155          bsf     INTCON,GIE      ;enable global interrupt bit
                 00156
002D             00157 start
002D 1C24        00158          btfss   flags,BEGIN     ;wait for a start measurement key press
002E 282D        00159          goto    start
002F 1024        00160          bcf     flags,BEGIN     ;clear start measurement flag
                 00161
0030 138B        00162          bcf     INTCON,GIE      ;transmit a start measurement message
0031 3053        00163          movlw   'S'             ;to the PC
0032 20AD        00164          call    Send
0033 178B        00165          bsf     INTCON,GIE
                 00166
0034 01C2        00167          clrf    TimeHI          ;reset Time registers
0035 01C1        00168          clrf    TimeMID
0036 01C0        00169          clrf    TimeLO
0037 1E86        00170          btfss   PORTB,WHICH     ;detect if resistor or capacitor measure
0038 2862        00171          goto    Capacitor
                 00172
0039             00173 Resistor
0039 1683        00174          bsf     STATUS,RP0      ;set V0 to input
003A 1406        00175          bsf     TRISB,V0
003B 1283        00176          bcf     STATUS,RP0
003C 20FB        00177          call    AnalogOn        ;turn analog on
003D 0181        00178          clrf    TMR0
003E 0000        00179          nop
003F 1586        00180          bsf     PORTB,V3        ;turn power on to unknown RC ckt
0040 19A4        00181 RwaitU   btfsc   flags,F_ERROR   ;detect if an error occurs
0041 288B        00182          goto    ErrorDetect
0042 1FA4        00183          btfss   flags,DONE      ;measurement completed flag
0043 2840        00184          goto    RwaitU
```


© 1997 Microchip Technology Inc.

```
0044 13A4    00185          bcf     flags,DONE      ;clear measurement completed flag
0045 2111    00186          call    AnalogOff       ;turn analog off
             00187
0046 2126    00188          call    SwapTtoA        ;swap Time to accumulator a
0047 3007    00189          movlw   RkHI            ;swap known resistance value
0048 00D3    00190          movwf   ACCbHI          ;to accumulator b
0049 309D    00191          movlw   RkMID
004A 00D4    00192          movwf   ACCbMID
004B 3038    00193          movlw   RkLO
004C 00D5    00194          movwf   ACCbLO
004D 2230    00195          call    Mpy24           ;multiply accumulator a and b
             00196
004E 1683    00197          bsf     STATUS,RP0      ;set V3 to input
004F 1586    00198          bsf     TRISB,V3
0050 1283    00199          bcf     STATUS,RP0
0051 20FB    00200          call    AnalogOn        ;turn analog on
0052 0181    00201          clrf    TMR0
0053 0000    00202          nop
0054 1406    00203          bsf     PORTB,V0        ;turn power on to known RC ckt
0055 19A4    00204 RwaitK   btfsc   flags,F_ERROR   ;detect if an error occurs
0056 288B    00205          goto    ErrorDetect
0057 1FA4    00206          btfss   flags,DONE      ;measurement completed flag
0058 2855    00207          goto    RwaitK
0059 13A4    00208          bcf     flags,DONE      ;clear measurement completed flag
005A 2111    00209          call    AnalogOff       ;turn analog off
             00210
005B 2126    00211          call    SwapTtoA        ;swap Time to accumulator a
005C 224B    00212          call    Div24           ;divide multiply by known time
             00213
005D 138B    00214          bcf     INTCON,GIE      ;disable all interrupts
005E 3052    00215          movlw   'R'             ;transmit, for R measurement
005F 20AD    00216          call    Send
0060 178B    00217          bsf     INTCON,GIE      ;enable global interrupt bit
0061 282D    00218          goto    start           ;restart
             00219
0062         00220 Capacitor
0062 1683    00221          bsf     STATUS,RP0      ;set V2 to input
0063 1506    00222          bsf     TRISB,V2
0064 1283    00223          bcf     STATUS,RP0
0065 20FB    00224          call    AnalogOn        ;turn analog on
0066 0181    00225          clrf    TMR0
0067 0000    00226          nop
0068 1486    00227          bsf     PORTB,V1        ;turn power on to unknown RC ckt
0069 19A4    00228 CwaitU   btfsc   flags,F_ERROR   ;detect if an error occurs
006A 288B    00229          goto    ErrorDetect
006B 1FA4    00230          btfss   flags,DONE      ;measurement completed flag
006C 2869    00231          goto    CwaitU
006D 13A4    00232          bcf     flags,DONE      ;clear measurement completed flag
006E 2111    00233          call    AnalogOff       ;turn analog off
             00234
006F 2126    00235          call    SwapTtoA        ;swap Time to accumulator a
0070 3007    00236          movlw   CkHI            ;swap known resistance value
0071 00D3    00237          movwf   ACCbHI          ;to accumulator b
0072 3047    00238          movlw   CkMID
0073 00D4    00239          movwf   ACCbMID
0074 3048    00240          movlw   CkLO
0075 00D5    00241          movwf   ACCbLO
0076 2230    00242          call    Mpy24           ;multiply accumulator a and b
             00243
0077 1683    00244          bsf     STATUS,RP0      ;set V3 to input
0078 1606    00245          bsf     TRISB,V4
0079 1283    00246          bcf     STATUS,RP0
007A 20FB    00247          call    AnalogOn        ;turn analog on
007B 0181    00248          clrf    TMR0
007C 0000    00249          nop
007D 1486    00250          bsf     PORTB,V1        ;turn power on to known RC ckt
```


© 1997 Microchip Technology Inc.

```
007E 19A4    00251 CwaitK  btfsc   flags,F_ERROR   ;detect if an error occurs
007F 288B    00252         goto    ErrorDetect
0080 1FA4    00253         btfss   flags,DONE      ;measurement completed flag
0081 287E    00254         goto    CwaitK
0082 13A4    00255         bcf     flags,DONE      ;clear measurement completed flag
0083 2111    00256         call    AnalogOff       ;turn analog off
             00257
0084 2126    00258         call    SwapTtoA        ;swap Time to accumulator a
0085 224B    00259         call    Div24           ;divide multiply by known time
             00260
0086 138B    00261         bcf     INTCON,GIE      ;disable all interrupts
0087 3043    00262         movlw   'C'             ;transmit, for C measurement
0088 20AD    00263         call    Send
0089 178B    00264         bsf     INTCON,GIE      ;enable global interrupt bit
008A 282D    00265         goto    start           ;restart
             00266
008B         00267 ErrorDetect
008B 1283    00268         bcf     STATUS,RP0      ;disable TMR0
008C 128B    00269         bcf     INTCON,T0IE
008D 110B    00270         bcf     INTCON,T0IF
008E 2111    00271         call    AnalogOff       ;turn analog off
008F 11A4    00272         bcf     flags,F_ERROR   ;clear error flag
             00273
0090 138B    00274         bcf     INTCON,GIE      ;disable all interrupts
0091 3045    00275         movlw   'E'             ;transmit, for C measurement
0092 20AD    00276         call    Send
0093 178B    00277         bsf     INTCON,GIE      ;enable global interrupt bit
0094 282D    00278         goto    start           ;restart
             00279
             00280 ;*************************************************************************
             00281 ;*-----------------------------------------------------------------------*
             00282 ;*-  RS232 Transmit Routine                                             -*
             00283 ;*-  Borrowed from AN593, "Serial Port Routines Without Using Timer0"  -*
             00284 ;*-  Author: Stan D'Souza                                               -*
             00285 ;*-  This is the routine that interfaces directly to the hardware       -*
             00286 ;*-----------------------------------------------------------------------*
             00287 ;*************************************************************************
0095         00288 Transmit
0095 1283    00289         bcf     STATUS,RP0
0096 00A6    00290         movwf   txreg
0097 1386    00291         bcf     tx              ;send start bit
0098 3020    00292         movlw   baudconst
0099 00A7    00293         movwf   delay
009A 3009    00294         movlw   0x9
009B 00A5    00295         movwf   count
009C         00296 txbaudwait
009C 0BA7    00297         decfsz  delay, F
009D 289C    00298         goto    txbaudwait
009E 3020    00299         movlw   baudconst
009F 00A7    00300         movwf   delay
00A0 0BA5    00301         decfsz  count, F
00A1 28A6    00302         goto    SendNextBit
00A2 3009    00303         movlw   0x9
00A3 00A5    00304         movwf   count
00A4 1786    00305         bsf     tx              ;send stop bit
00A5 0008    00306         return
00A6         00307 SendNextBit
00A6 0CA6    00308         rrf     txreg, F
00A7 1C03    00309         btfss   STATUS,C
00A8 28AB    00310         goto    Setlo
00A9 1786    00311         bsf     tx
00AA 289C    00312         goto    txbaudwait
00AB 1386    00313 Setlo   bcf     tx
00AC 289C    00314         goto    txbaudwait
             00315 ;_______________________________________________________________________
             00316
```

```
                00317 ;*************************************************************************
                00318 ;*-----------------------------------------------------------------------*
                00319 ;*-     Generic Transmit Routine                                        -*
                00320 ;*-     Sends what is currently in the W register and accumulator ACCc -*
                00321 ;*-----------------------------------------------------------------------*
                00322 ;*************************************************************************
00AD            00323 Send
00AD 2095       00324         call    Transmit
00AE 2146       00325         call    delay1          ;delay between bytes
00AF 0856       00326         movf    ACCcHI,W        ;transmit high resistance byte
00B0 2095       00327         call    Transmit
00B1 2146       00328         call    delay1          ;delay between bytes
00B2 0857       00329         movf    ACCcMID,W       ;transmit mid resistance byte
00B3 2095       00330         call    Transmit
00B4 2146       00331         call    delay1          ;delay between bytes
00B5 0858       00332         movf    ACCcLO,W        ;transmit low resistance byte
00B6 2095       00333         call    Transmit
00B7 2146       00334         call    delay1          ;delay between bytes
00B8 0008       00335         return
                00336 ;_____________________________________________________________________
                00337
                00338 ;*************************************************************************
                00339 ;*-----------------------------------------------------------------------*
                00340 ;*-     Interrupt Service Routines                                      -*
                00341 ;*-----------------------------------------------------------------------*
                00342 ;*************************************************************************
00B9            00343 ServiceInterrupts
00B9 00A0       00344         movwf   W_TEMP              ;Pseudo push instructions
00BA 0E03       00345         swapf   STATUS,W
00BB 1283       00346         bcf     STATUS,RP0
00BC 00A1       00347         movwf   STATUS_TEMP
                00348
00BD 0801       00349         movf    TMR0,W
00BE 00A3       00350         movwf   Ttemp
00BF 190B       00351         btfsc   INTCON,T0IF         ;Service Timer0 overflow
00C0 20E5       00352         call    ServiceTimer
00C1 1B0C       00353         btfsc   PIR1,CMIF           ;Stops Timer0, Records Value
00C2 20EC       00354         call    ServiceComparator
00C3 180B       00355         btfsc   INTCON,RBIF         ;Service pushbutton switch
00C4 20CB       00356         call    ServiceKeystroke    ;Starts a measurement
                00357
00C5 1283       00358         bcf     STATUS,RP0
00C6 0E21       00359         swapf   STATUS_TEMP,W       ;Pseudo pop instructions
00C7 0083       00360         movwf   STATUS
00C8 0EA0       00361         swapf   W_TEMP,F
00C9 0E20       00362         swapf   W_TEMP,W
                00363
00CA 0009       00364         retfie
                00365 ;_____________________________________________________________________
                00366
                00367 ;*************************************************************************
                00368 ;*-----------------------------------------------------------------------*
                00369 ;*-     Borrowed from AN552, "Implementing Wake-up on Key Stroke"       -*
                00370 ;*-     Author: Stan D'Souza                                            -*
                00371 ;*-----------------------------------------------------------------------*
                00372 ;*************************************************************************
00CB            00373 ServiceKeystroke
00CB 118B       00374         bcf     INTCON,RBIE     ;disable interrupt
00CC 0906       00375         comf    PORTB,W         ;read PORTB
00CD 100B       00376         bcf     INTCON,RBIF     ;clear interrupt flag
00CE 3940       00377         andlw   B'01000000'
00CF 1903       00378         btfsc   STATUS,Z
00D0 28D6       00379         goto    NotSwitch
00D1 2143       00380         call    delay16         ;de-bounce switch for 16msec
00D2 0906       00381         comf    PORTB,W         ;read PORTB again
00D3 20D9       00382         call    KeyRelease      ;check for key release
```


© 1997 Microchip Technology Inc.

```
00D4 1424    00383          bsf     flags,BEGIN
00D5 0008    00384          return
             00385
00D6         00386 NotSwitch                        ;detected other PORTB pin change
00D6 100B    00387          bcf     INTCON,RBIF     ;reset RBI interrupt
00D7 158B    00388          bsf     INTCON,RBIE
00D8 0008    00389          return
             00390
00D9         00391 KeyRelease
00D9 2143    00392          call    delay16         ;debounce switch
00DA 0906    00393          comf    PORTB,W         ;read PORTB
00DB 100B    00394          bcf     INTCON,RBIF     ;clear flag
00DC 158B    00395          bsf     INTCON,RBIE     ;enable interrupt
00DD 3940    00396          andlw   B'01000000'
00DE 1903    00397          btfsc   STATUS,Z        ;key still pressed?
00DF 0008    00398          return                  ;if no, then return
00E0 0063    00399          sleep                   ;else, save power
00E1 118B    00400          bcf     INTCON,RBIE     ;disable interrupts
00E2 0906    00401          comf    PORTB,W         ;read PORTB
00E3 100B    00402          bcf     INTCON,RBIF     ;clear flag
00E4 28D9    00403          goto    KeyRelease      ;try again
             00404 ;_____________________________________________________________________
             00405
             00406 ;*****************************************************************
             00407 ;*-------------------------------------------------------------*
             00408 ;*-     ISR to service a Timer0 overflow                      -*
             00409 ;*-------------------------------------------------------------*
             00410 ;*****************************************************************
00E5         00411 ServiceTimer
00E5 0AC1    00412          incf    TimeMID,F       ;increment middle Time byte
00E6 1903    00413          btfsc   STATUS,Z        ;if middle overflows,
00E7 0AC2    00414          incf    TimeHI,F        ;increment high Time byte
00E8 1AC2    00415          btfsc   TimeHI,EMPTY    ;check if component is connected
00E9 15A4    00416          bsf     flags,F_ERROR   ;set error flag
00EA 110B    00417          bcf     INTCON,T0IF     ;clear TMR0 interrupt flag
00EB 0008    00418          return
             00419 ;_____________________________________________________________________
             00420
             00421 ;*****************************************************************
             00422 ;*-------------------------------------------------------------*
             00423 ;*-     ISR to service a Comparator interrupt                 -*
             00424 ;*-------------------------------------------------------------*
             00425 ;*****************************************************************
00EC         00426 ServiceComparator
00EC 1283    00427          bcf     STATUS,RP0      ;select bank 0
00ED 1E86    00428          btfss   PORTB,WHICH     ;detect which measurement, R or C?
00EE 28F2    00429          goto    capcomp
00EF 1F1F    00430          btfss   CMCON,C1OUT     ;detect if R ckt has interrupted
00F0 28F4    00431          goto    scstop
00F1 28F9    00432          goto    scend
00F2         00433 capcomp
00F2 1B9F    00434          btfsc   CMCON,C2OUT     ;detect if C ckt has interrupted
00F3 28F9    00435          goto    scend
00F4         00436 scstop
00F4 128B    00437          bcf     INTCON,T0IE     ;disable TMR0 interrupts
00F5 110B    00438          bcf     INTCON,T0IF
00F6 0823    00439          movf    Ttemp,W
00F7 00C0    00440          movwf   TimeLO
00F8 17A4    00441          bsf     flags,DONE      ;set DONE flag
00F9         00442 scend
00F9 130C    00443          bcf     PIR1,CMIF       ;clear comparator interrupt flag
00FA 0008    00444          return
             00445 ;_____________________________________________________________________
             00446
             00447 ;*****************************************************************
             00448 ;*-------------------------------------------------------------*
```


© 1997 Microchip Technology Inc.

```
                00449 ;*-     Turn Comparators and Vref On                                -*
                00450 ;*-------------------------------------------------------------------*
                00451 ;*******************************************************************
00FB            00452 AnalogOn
00FB 1283       00453         bcf     STATUS,RP0      ;select bank 0
00FC 3002       00454         movlw   0x02            ;turn comparators on, mode 010
00FD 009F       00455         movwf   CMCON           ;4 inputs multiplexed to 2 comparators
00FE 1683       00456         bsf     STATUS,RP0      ;select bank 1
00FF 300F       00457         movlw   0x0F            ;make PORTA<3:0> all inputs
0100 0085       00458         movwf   TRISA
0101 179F       00459         bsf     VRCON,VREN
0102 1283       00460         bcf     STATUS,RP0      ;select bank 0
0103 2140       00461         call    delay20         ;20msec delay
0104 089F       00462         movf    CMCON,F         ;clear comparator mismatch condition
0105 130C       00463         bcf     PIR1,CMIF       ;clear comparator interrupt flag
0106 1683       00464         bsf     STATUS,RP0
0107 170C       00465         bsf     PIE1,CMIE       ;enable comparator interrupts
0108 1283       00466         bcf     STATUS,RP0
0109 170B       00467         bsf     INTCON,PEIE     ;enable peripheral interrupts
010A 11A4       00468         bcf     flags,F_ERROR
010B 0181       00469         clrf    TMR0            ;clear TMR0 counter
010C 0000       00470         nop
010D 0000       00471         nop
010E 110B       00472         bcf     INTCON,T0IF     ;clear TMR0 interrupt flag
010F 168B       00473         bsf     INTCON,T0IE     ;enable TMR0 interrupts
0110 0008       00474         return
                00475 ;___________________________________________________________________
                00476
                00477 ;*******************************************************************
                00478 ;*-------------------------------------------------------------------*
                00479 ;*-     Turn Comparators and Vref Off                              -*
                00480 ;*-------------------------------------------------------------------*
                00481 ;*******************************************************************
0111            00482 AnalogOff
0111 1283       00483         bcf     STATUS,RP0
0112 130B       00484         bcf     INTCON,PEIE
0113 3080       00485         movlw   0x80            ;reset PORTB value
0114 0086       00486         movwf   PORTB
0115 1683       00487         bsf     STATUS,RP0      ;select bank 1
0116 130C       00488         bcf     PIE1,CMIE       ;disable comparator interrupts
0117 0185       00489         clrf    TRISA           ;set PORTA pins to outputs, discharge RC ckt
0118 3060       00490         movlw   0x60            ;set PORTB 7,4-0 as outputs, 6,5 as inputs
0119 0086       00491         movwf   TRISB
011A 139F       00492         bcf     VRCON,VREN      ;disable Vref
011B 1283       00493         bcf     STATUS,RP0      ;select bank 0
011C 3007       00494         movlw   0x07
011D 009F       00495         movwf   CMCON           ;disable comparators
011E 2140       00496         call    delay20         ;20msec delay
011F 089F       00497         movf    CMCON,F         ;clear comparator mismatch condition
0120 130C       00498         bcf     PIR1,CMIF       ;clear comparator interrupt flag
0121 110B       00499         bcf     INTCON,T0IF     ;clear Timer0 interrupt flag
0122 213D       00500         call    vlong           ;long delay to allow capacitors to discharge
0123 213D       00501         call    vlong
0124 213D       00502         call    vlong
0125 0008       00503         return
                00504 ;___________________________________________________________________
                00505
                00506 ;*******************************************************************
                00507 ;*-------------------------------------------------------------------*
                00508 ;*-     Swap Time to Accumulator a                                 -*
                00509 ;*-------------------------------------------------------------------*
                00510 ;*******************************************************************
0126            00511 SwapTtoA
0126 1283       00512         bcf     STATUS,RP0
0127 0842       00513         movf    TimeHI,W
0128 00D0       00514         movwf   ACCaHI
```


© 1997 Microchip Technology Inc.

```
0129 0841    00515          movf    TimeMID,W
012A 00D1    00516          movwf   ACCaMID
012B 0840    00517          movf    TimeLO,W
012C 00D2    00518          movwf   ACCaLO
012D 01C2    00519          clrf    TimeHI
012E 01C1    00520          clrf    TimeMID
012F 01C0    00521          clrf    TimeLO
0130 0008    00522          return
             00523 ;_____________________________________________________________________
             00524
             00525 ;*********************************************************************
             00526 ;*-------------------------------------------------------------------*
             00527 ;*-     Transmit the Boot Message                                   -*
             00528 ;*-------------------------------------------------------------------*
             00529 ;*********************************************************************
0131         00530 BootMSG
0131 1283    00531          bcf     STATUS,RP0      ;select bank 0
0132 3002    00532 msg      movlw   HIGH Table      ;init the PCH for a table call
0133 008A    00533          movwf   PCLATH
0134 0828    00534          movf    offset,W        ;move table offset into W
0135 2200    00535          call    Table           ;get table value
0136 2095    00536          call    Transmit        ;transmit table value
0137 2146    00537          call    delay1          ;delay between bytes
0138 0BA8    00538          decfsz  offset,F        ;check for end of table
0139 2932    00539          goto    msg
013A 3010    00540          movlw   0x10            ;reset table offset
013B 00A8    00541          movwf   offset
013C 0008    00542          return
             00543 ;_____________________________________________________________________
             00544
             00545 ;*********************************************************************
             00546 ;*-------------------------------------------------------------------*
             00547 ;*-     Delay Routines                                              -*
             00548 ;*-------------------------------------------------------------------*
             00549 ;*********************************************************************
013D 30FF    00550 vlong    movlw   0xff            ;very long delay, approx 200msec
013E 00A9    00551          movwf   msb
013F 2948    00552          goto    d1
0140         00553 delay20                          ;20 msec delay
0140 301A    00554          movlw   .26
0141 00A9    00555          movwf   msb
0142 2948    00556          goto    d1
0143         00557 delay16                          ;16 msec delay
0143 3015    00558          movlw   .21
0144 00A9    00559          movwf   msb
0145 2948    00560          goto    d1
0146         00561 delay1                           ;approx 750nsec delay
0146 3001    00562          movlw   .1
0147 00A9    00563          movwf   msb
0148 30FF    00564 d1       movlw   0xff
0149 00AA    00565          movwf   lsb
014A 0BAA    00566 d2       decfsz  lsb,F
014B 294A    00567          goto    d2
014C 0BA9    00568          decfsz  msb,F
014D 2948    00569          goto    d1
014E 0008    00570          return
             00571 ;_____________________________________________________________________
             00572
             00573
0200         00574          org     0x200
             00575
             00576
             00577 ;*********************************************************************
             00578 ;*-------------------------------------------------------------------*
             00579 ;*-     Table for Boot Message                                      -*
             00580 ;*-------------------------------------------------------------------*
```


© 1997 Microchip Technology Inc.

```
                00581 ;*******************************************************************
0200            00582 Table                             ;boot message "PICMETER Booted!"
0200 0782       00583         addwf   PCL, F            ;add W to PCL
0201 3400       00584         retlw   0
0202 3421       00585         retlw   '!'
0203 3464       00586         retlw   'd'
0204 3465       00587         retlw   'e'
0205 3474       00588         retlw   't'
0206 346F       00589         retlw   'o'
0207 346F       00590         retlw   'o'
0208 3442       00591         retlw   'B'
0209 3420       00592         retlw   ' '
020A 3452       00593         retlw   'R'
020B 3445       00594         retlw   'E'
020C 3454       00595         retlw   'T'
020D 3445       00596         retlw   'E'
020E 344D       00597         retlw   'M'
020F 3443       00598         retlw   'C'
0210 3449       00599         retlw   'I'
0211 3450       00600         retlw   'P'
                00601 ;_____________________________________________________________________
                00602
                00603 ;*******************************************************************
                00604 ;*-------------------------------------------------------------------*
                00605 ;*-     24-bit Addition                                             -*
                00606 ;*-                                                                 -*
                00607 ;*-     Uses ACCa and ACCb                                          -*
                00608 ;*-                                                                 -*
                00609 ;*-     ACCa + ACCb -> ACCb                                         -*
                00610 ;*-------------------------------------------------------------------*
                00611 ;*******************************************************************
0212            00612 Add24
0212 0852       00613         movf    ACCaLO,W
0213 07D5       00614         addwf   ACCbLO, F         ;add low bytes
0214 1803       00615         btfsc   STATUS,C          ;add in carry if necessary
0215 2A1D       00616         goto    A2
0216 0851       00617 A1      movf    ACCaMID,W
0217 07D4       00618         addwf   ACCbMID, F        ;add mid bytes
0218 1803       00619         btfsc   STATUS,C          ;add in carry if necessary
0219 0AD3       00620         incf    ACCbHI, F
021A 0850       00621         movf    ACCaHI,W
021B 07D3       00622         addwf   ACCbHI, F         ;add high bytes
021C 3400       00623         retlw   0
021D 0AD4       00624 A2      incf    ACCbMID, F
021E 1903       00625         btfsc   STATUS,Z
021F 0AD3       00626         incf    ACCbHI, F
0220 2A16       00627         goto    A1
                00628 ;_____________________________________________________________________
                00629
                00630 ;*******************************************************************
                00631 ;*-------------------------------------------------------------------*
                00632 ;*-     Subtraction ( 24 - 24 -> 24 )                                 -*
                00633 ;*-                                                                   -*
                00634 ;*-     Uses ACCa, ACCb, ACCd                                         -*
                00635 ;*-                                                                   -*
                00636 ;*-     ACCa -> ACCd,                                                 -*
                00637 ;*-     2's complement ACCa,                                          -*
                00638 ;*-     call Add24 ( ACCa + ACCb -> ACCb ),                           -*
                00639 ;*-     ACCd -> ACCa                                                  -*
                00640 ;*-------------------------------------------------------------------*
                00641 ;*******************************************************************
0221            00642 Sub24
0221 0850       00643         movf    ACCaHI,W                ;Transfer ACCa to ACCd
0222 00D9       00644         movwf   ACCdHI
0223 0851       00645         movf    ACCaMID,W
0224 00DA       00646         movwf   ACCdMID
```


© 1997 Microchip Technology Inc.

```
0225 0852    00647          movf    ACCaLO,W
0226 00DB    00648          movwf   ACCdLO
0227 2275    00649          call    compA                   ;2's complement ACCa
0228 2212    00650          call    Add24                   ;Add ACCa to ACCb
0229 0859    00651          movf    ACCdHI,W                ;Transfer ACCd to ACCa
022A 00D0    00652          movwf   ACCaHI
022B 085A    00653          movf    ACCdMID,W
022C 00D1    00654          movwf   ACCaMID
022D 085B    00655          movf    ACCdLO,W
022E 00D2    00656          movwf   ACCaLO
022F 3400    00657          retlw   0
             00658 ;_____________________________________________________________________
             00659
             00660 ;*********************************************************************
             00661 ;*-------------------------------------------------------------------*
             00662 ;*-     Multiply ( 24 X 24 -> 56 )                                  -*
             00663 ;*-                                                                 -*
             00664 ;*-     Uses ACCa, ACCb, ACCc, ACCd                                 -*
             00665 ;*-                                                                 -*
             00666 ;*-     ACCa * ACCb -> ACCb,ACCc  56-bit output                     -*
             00667 ;*-     with ACCb (ACCbHI,ACCbMID,ACCbLO) with 24 msb's and         -*
             00668 ;*-     ACCc (ACCcHI,ACCcMID,ACCcLO) with 24 lsb's                  -*
             00669 ;*-------------------------------------------------------------------*
             00670 ;*********************************************************************
0230         00671 Mpy24
0230 223F    00672          call    Msetup
0231 0CD9    00673 mloop    rrf     ACCdHI, F       ;rotate d right
0232 0CDA    00674          rrf     ACCdMID, F
0233 0CDB    00675          rrf     ACCdLO, F
0234 1803    00676          btfsc   STATUS,C        ;need to add?
0235 2212    00677          call    Add24
0236 0CD3    00678          rrf     ACCbHI, F
0237 0CD4    00679          rrf     ACCbMID, F
0238 0CD5    00680          rrf     ACCbLO, F
0239 0CD6    00681          rrf     ACCcHI, F
023A 0CD7    00682          rrf     ACCcMID, F
023B 0CD8    00683          rrf     ACCcLO, F
023C 0BDC    00684          decfsz  temp, F         ;loop until all bits checked
023D 2A31    00685          goto    mloop
023E 3400    00686          retlw   0
             00687
023F         00688 Msetup
023F 3018    00689          movlw   0x18            ;for 24 bit shifts
0240 00DC    00690          movwf   temp
0241 0853    00691          movf    ACCbHI,W        ;move ACCb to ACCd
0242 00D9    00692          movwf   ACCdHI
0243 0854    00693          movf    ACCbMID,W
0244 00DA    00694          movwf   ACCdMID
0245 0855    00695          movf    ACCbLO,W
0246 00DB    00696          movwf   ACCdLO
0247 01D3    00697          clrf    ACCbHI
0248 01D4    00698          clrf    ACCbMID
0249 01D5    00699          clrf    ACCbLO
024A 3400    00700          retlw   0
             00701 ;_____________________________________________________________________
             00702
             00703 ;*********************************************************************
             00704 ;*-------------------------------------------------------------------*
             00705 ;*-     Division ( 56 / 24 -> 24 )                                  -*
             00706 ;*-                                                                 -*
             00707 ;*-     Uses ACCa, ACCb, ACCc, ACCd                                 -*
             00708 ;*-                                                                 -*
             00709 ;*-  56-bit dividend in ACCb,ACCc ( ACCb has msb's and ACCc has lsb's)  -*
             00710 ;*-     24-bit divisor in ACCa                                      -*
             00711 ;*-     quotient is stored in ACCc                                  -*
             00712 ;*-     remainder is stored in ACCb                                 -*
```


© 1997 Microchip Technology Inc.

```
                00713 ;*-------------------------------------------------------------------*
                00714 ;*********************************************************************
024B            00715 Div24
024B 2272       00716         call    Dsetup
                00717
024C 1003       00718 dloop   bcf     STATUS,C
024D 0DD8       00719         rlf     ACCcLO, F       ;Rotate dividend left 1 bit position
024E 0DD7       00720         rlf     ACCcMID, F
024F 0DD6       00721         rlf     ACCcHI, F
0250 0DD5       00722         rlf     ACCbLO, F
0251 0DD4       00723         rlf     ACCbMID, F
0252 0DD3       00724         rlf     ACCbHI, F
                00725
0253 1803       00726         btfsc   STATUS,C        ;invert carry and exclusive or with the
0254 2A58       00727         goto    clear           ;msb of the divisor then move this bit
0255 1FD0       00728         btfss   ACCaHI,msb_bit  ;into the lsb of the dividend
0256 0AD8       00729         incf    ACCcLO, F
0257 2A5A       00730         goto    cont
0258 1BD0       00731 clear   btfsc   ACCaHI,msb_bit
0259 0AD8       00732         incf    ACCcLO, F
                00733
025A 1858       00734 cont    btfsc   ACCcLO,lsb_bit  ;check the lsb of the dividend
025B 2A5E       00735         goto    minus
025C 2212       00736         call    Add24           ;if = 0, then add divisor to upper 24 bits
025D 2A5F       00737         goto    check           ;of dividend
025E 2221       00738 minus   call    Sub24           ;if = 1, then subtract divisor from upper
                00739                                 ;24 bits of dividend
                00740
025F 0BDC       00741 check   decfsz  temp,f          ;do 24 times
0260 2A4C       00742         goto    dloop
                00743
0261 1003       00744         bcf     STATUS,C
0262 0DD8       00745         rlf     ACCcLO, F       ;shift lower 24 bits of dividend 1 bit
0263 0DD7       00746         rlf     ACCcMID, F      ;position left
0264 0DD6       00747         rlf     ACCcHI, F
0265 1BD3       00748         btfsc   ACCbHI,msb_bit  ;exclusive or the inverse of the msb of the
0266 2A6A       00749         goto    w1              ;dividend with the msb of the divisor
0267 1FD0       00750         btfss   ACCaHI,msb_bit  ;store in the lsb of the dividend
0268 0AD8       00751         incf    ACCcLO, F
0269 2A6C       00752         goto    wzd
026A 1BD0       00753 w1      btfsc   ACCaHI,msb_bit
026B 0AD8       00754         incf    ACCcLO, F
026C 1FD3       00755 wzd     btfss   ACCbHI,msb_bit  ;if the msb of the remainder is set and
026D 2A71       00756         goto    wend
026E 1BD0       00757         btfsc   ACCaHI,msb_bit  ;the msb of the divisor is not
026F 2A71       00758         goto    wend
0270 2212       00759         call    Add24           ;add the divisor to the remainder to correct
                00760                                 ;for zero partial remainder
                00761
0271 3400       00762 wend    retlw   0               ;quotient in 24 lsb's of dividend
                00763                                 ;remainder in 24 msb's of dividend
                00764
0272            00765 Dsetup
0272 3018       00766         movlw   0x18            ;loop 24 times
0273 00DC       00767         movwf   temp
                00768
0274 3400       00769         retlw   0
                00770 ;_____________________________________________________________________
                00771
                00772 ;*********************************************************************
                00773 ;*-------------------------------------------------------------------*
                00774 ;*-     2's Complement                                              -*
                00775 ;*-                                                                 -*
                00776 ;*-     Uses ACCa                                                   -*
                00777 ;*-                                                                 -*
                00778 ;*-     Performs 2's complement conversion on ACCa                   -*
```


© 1997 Microchip Technology Inc.

```
                00779 ;*-------------------------------------------------------------------------*
                00780 ;*************************************************************************
0275            00781 compA
0275 09D2       00782         comf    ACCaLO, F       ;invert all bits in accumulator a
0276 09D1       00783         comf    ACCaMID, F
0277 09D0       00784         comf    ACCaHI, F
0278 0AD2       00785         incf    ACCaLO, F       ;add one to accumulator a
0279 1903       00786         btfsc   STATUS,Z
027A 0AD1       00787         incf    ACCaMID, F
027B 1903       00788         btfsc   STATUS,Z
027C 0AD0       00789         incf    ACCaHI, F
027D 3400       00790         retlw   0
                00791 ;_____________________________________________________________________________
                00792
                00793         END
PICMETER Firmware for PIC16C622
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---X----------- XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXX- ---------------- ---------------- ----------------
0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX--
2000 : -------X-------- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   447
Program Memory Words Free:  1601


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,    15 suppressed
```


© 1997 Microchip Technology Inc.

APPENDIX B: VISUAL BASIC PROGRAM

PICMTR.FRM

```
Sub Form_Load ()
    'Initialize the program
    Image1.Height = 600
    Image1.Width = 2700
    Frame1.Caption = "PICMETER Power Off"
    Label1.Caption = ""
    Label2.Caption = ""

    'Initialize Comm Port 1
    Comm1.RThreshold = 1
    Comm1.Handshaking = 0
    Comm1.Settings = "9600,n,8,1"
    Comm1.CommPort = 2
    Comm1.PortOpen = True

    'Initialize the global variable First%
    First% = 0
End Sub

Sub Form_Unload (Cancel As Integer)
    'Unload PICMETER
    Comm1.RTSEnable = False
    Comm1.DTREnable = False
    Comm1.PortOpen = False
    Unload PICMETER
End Sub

Sub Comm1_OnComm ()
    Dim Value As Double
    Dim High As Double
    Dim Medium As Double
    Dim Low As Double

    'Received a character
    If Comm1.CommEvent = 2 Then
        If First% = 0 Then
            If Comm1.InBufferCount = 16 Then
                Label1.FontSize = 10
                InString$ = Comm1.Input
                If InString$ = "PICMETER Booted!" Then
                    Frame1.Caption = "PICMETER Booted!"
                End If
                First% = 1
                Comm1.InputLen = 4
            End If
        Else
            If Comm1.InBufferCount >= 4 Then
                InString$ = Comm1.Input
                If Left$(InString$, 1) = "R" Then
                    Frame1.Caption = "Resistance"
                    Label2.FontName = "Symbol"
                    Label2.Caption = "KW"
                    Label1.FontSize = 24
                ElseIf Left$(InString$, 1) = "C" Then
                    Frame1.Caption = "Capacitance"
                    Label2.FontName = "MS Sans Serif"
                    Label2.Caption = "nF"
                    Label1.FontSize = 24
                ElseIf Left$(InString$, 1) = "E" Then
                    Frame1.Caption = "Error Detected"
                    Label2.Caption = ""
                ElseIf Left$(InString$, 1) = "S" Then
                    Frame1.Caption = "Measuring Component"
                    Label2.Caption = ""
                Else
                    Frame1.Caption = "Error Detected"
                    Label2.Caption = ""
                End If
```


© 1997 Microchip Technology Inc.

```
                If Frame1.Caption = "Error Detected" Then
                    Label1.Caption = ""
                ElseIf Frame1.Caption = "Measuring Component" Then
                    Label1.Caption = ""
                Else
                    High = 65536# * Asc(Mid$(InString$, 2, 1))
                    Medium = 256# * Asc(Mid$(InString$, 3, 1))
                    Low = Asc(Mid$(InString$, 4, 1))
                    Label1.Caption = Format$((High + Medium + Low) / 1000, "###0.0")
                End If
            End If
        End If
    End If
End Sub

Sub Check3D1_Click (Value As Integer)
    'Control Power to the PICMETER
    If Check3D1.Value = False Then
        Comm1.InputLen = 0
        Label1.Caption = ""
        Label2.Caption = ""
        Comm1.RTSEnable = False
        Comm1.DTREnable = False
        Frame1.Caption = "PICMETER Power Off"
        InString$ = Comm1.Input
    Else
        Frame1.Caption = ""
        First% = 0
        Comm1.InputLen = 0
        InString$ = Comm1.Input
        Comm1.RTSEnable = True
        Comm1.DTREnable = True
    End If
End Sub

Sub menExitTop_Click ()
    'Unload PICMETER
    Unload PICMETER
End Sub

Sub Option1_Click ()
    'Open COM1 for communications
    If Option1.Value = True Then
        If Comm1.CommPort = 2 Then
            Comm1.PortOpen = False
            Comm1.CommPort = 1
            Comm1.PortOpen = True
        End If
    End If
End Sub

Sub Option2_Click ()
    'Open COM2 for communications
    If Option2.Value = True Then
        If Comm1.CommPort = 1 Then
            Comm1.PortOpen = False
            Comm1.CommPort = 2
            Comm1.PortOpen = True
        End If
    End If
End Sub
```

PICMETER.BAS

```
Global I%
Global First%
```


© 1997 Microchip Technology Inc.

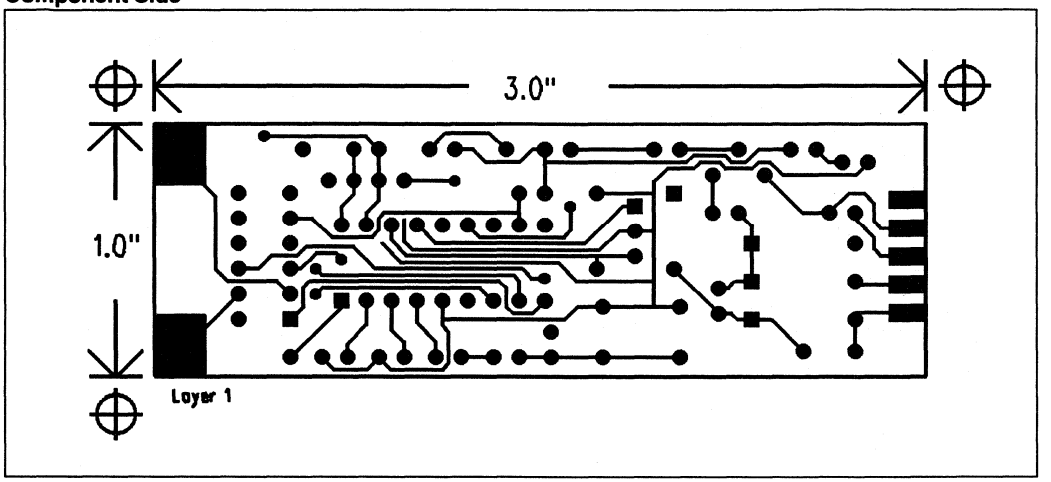
APPENDIX C: PICMETER PCB LAYOUT

Boards Manufactured by: Southwest Circuits

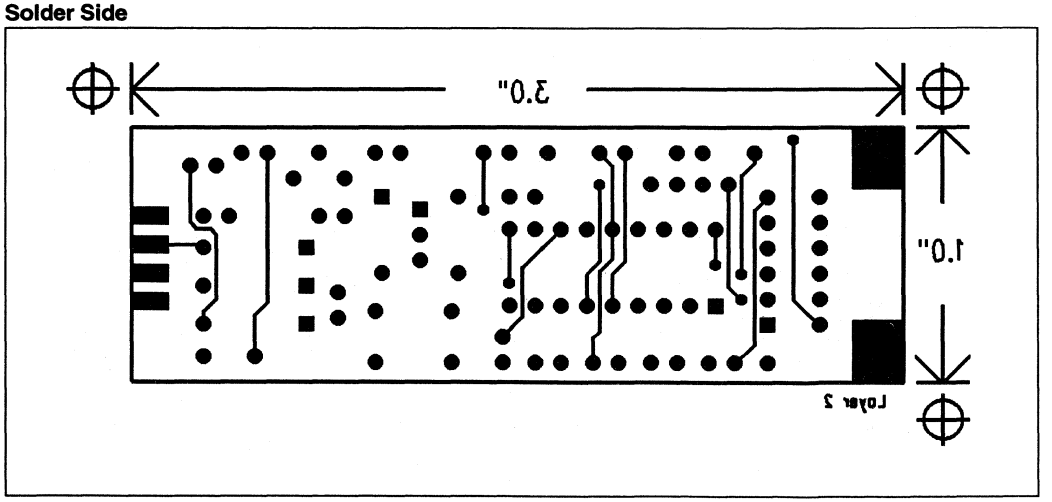
Contact: Perry Groves
3760 E. 43rd Place
Tucson, AZ 85713
1-520-745-8515

The following artwork is not printed to scale:

Component Side

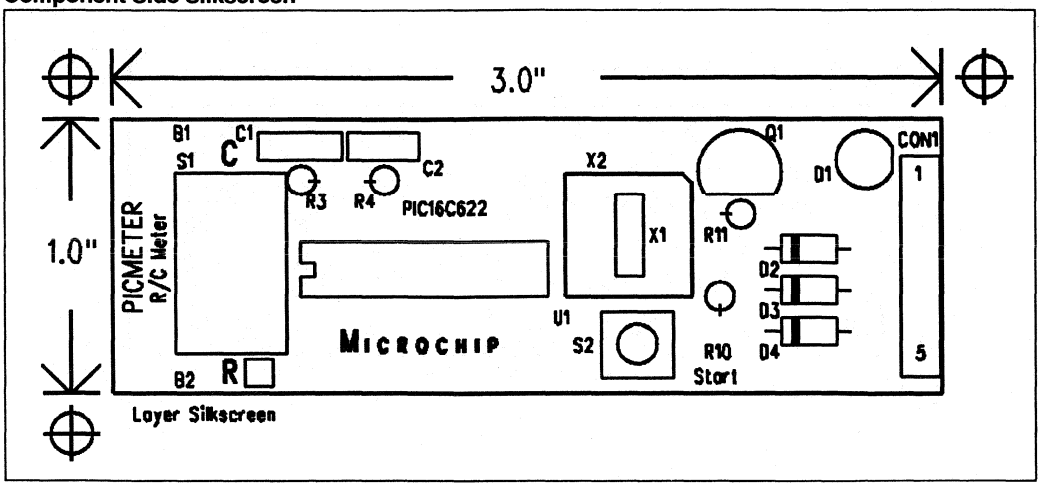


Solder Side

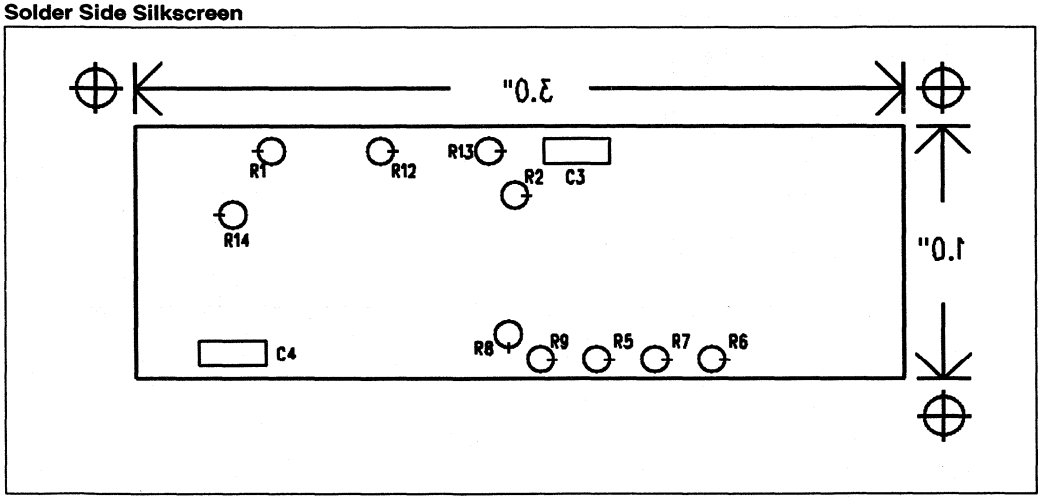


© 1997 Microchip Technology Inc.

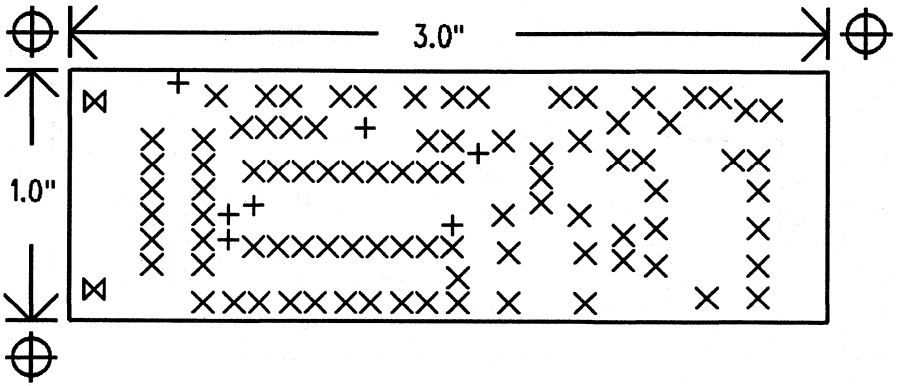
Component Side Silkscreen



Solder Side Silkscreen



Manufacturing Drawing



SIZE	QTY	SYM
18	7	+
37	89	X
95	2	⋈

© 1997 Microchip Technology Inc.

Digital Signal Processing with the PIC16C74

Author: *Darius Mostowfi*
Design Consultant

INTRODUCTION

This application note describes the basic issues that need to be addressed in order to implement digital signal processing systems using the PIC16C74 and provides application code modules and examples for DTMF tone generation, a 60 Hz notch filter, and a simple PID compensator for control systems. These routines can also be used with other PIC16C6X and PIC16C7XXX processors with minor modifications and the addition of external analog I/O devices.

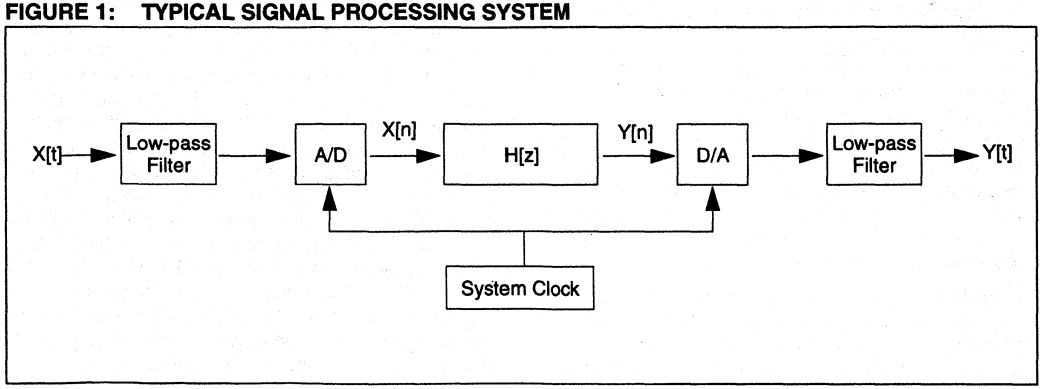
The use of general purpose microcontrollers for low-end digital signal processing applications has become more commonplace these days with the availability of higher speed processors. Since most signal processing systems consist of a host processor and dedicated DSP chip, the use of a single microcontroller to perform both these functions provides a simpler and lower cost solution. In addition, the single chip design will consume less power which is ideal for battery powered applications. The PIC16C74 with its on-chip A/D, PWM module, and fast CPU is an ideal candidate for use in these low-bandwidth signal processing applications.

A typical signal processing system includes an A/D converter, D/A converter, and CPU that performs the signal processing algorithm as shown in Figure 1.

The input signal, *x(t)*, is first passed through an input filter (commonly called the anti-aliasing filter) whose function is to bandlimit the signal to below the Nyquist rate (one half the sampling frequency) to prevent aliasing. The signal is then digitized by the A/D converter at a rate determined by the sample clock to produce *x(n)*, the discrete-time input sequence. The system transfer function, *H(z)*, is typically implemented in the time-domain using a difference equation. The output sample, *y(n)*, is then converted back into the continuous-time signal, *y(t)*, by the D/A converter and output low-pass filter.

The calculation of the output signal using a difference equation requires a multiply and accumulate (MAC) operation. This is typically a single-cycle instruction on DSP chips but can take many cycles to perform on a standard microcontroller since it must be implemented in code. Since the digitization of the signal, calculation of the output, and output to the D/A converter all must be completed within the sample clock period, the speed at which this can be done determines the maximum bandwidth that can be achieved with the system. The relatively slow speed of most microcontrollers is the major limitation when they are used in DSP applications but the PIC16C74's fast instruction execution speed (as fast as 200 ns/instruction) can provide the performance required to implement relatively low bandwidth systems. In addition, the device's on-chip A/D and PWM modules provide all the functions needed for a single chip system. Only a few external components are needed to use the PIC16C74 for tone generation, filtering of transducer signals, or low bandwidth control.

FIGURE 1: TYPICAL SIGNAL PROCESSING SYSTEM



© 1997 Microchip Technology Inc.

CODE DEVELOPMENT TOOLS

The code for these applications was written using Byte Craft's MPC C compiler. The MPC compiler provides an Integrated Development Environment (IDE) and generates highly optimized code for the entire PICmicro™ family. For new PICmicro users that are familiar with C, this is an ideal way to quickly develop code for these processors. In addition, the listing files can be studied in order to learn the details of PICmicro assembly language. The modules and examples for this application note use C for the main program body and in-line assembly language for the time-critical routines. MPC provides interrupt support so that interrupt service routines (ISRs) can be easily written in either C or assembly. This feature was used to provide a timer ISR for one of the code modules. The compiler proved to be a valuable tool that allowed both high level and assembly language routines to be written and tested quickly.

In order to provide the double precision math functions required for this application note, a couple of existing math functions written for the PIC16C54 (AN525, *Programming PIC16C5X Devices on Logical Devices*) were converted for use with MPC. The double precision multiply and addition routines were modified by first changing all RAM declarations done in EQU statements to C "*unsigned char*" variable declarations. The main body of assembly language code was preceded by "`#asm`" and ended by "`#endasm`" preprocessor directives which tell the compiler where the in-line assembly code starts and ends. Finally, any macro sections and register names that are defined differently in MPC were changed.

The assembly language routines for tone generation and filtering were also written as C functions using the compiler. Assembly language routines written in this way can be called directly from other assembly language modules or called directly from C by using the label name as a C function. Source listings for all the modules and example programs can be found in the appendices at the end of this application note. These modules can be directly compiled using the MPC compiler or, alternatively, the assembly language sections can be used with MPASM with minor modifications.

Number Representation and Math Routines

One of the challenges of using any general purpose microcontroller for signal processing algorithms is in implementing the finite word-length arithmetic required to perform the calculations. As mentioned before, the speed at which the MAC operations can be performed limits the maximum achievable bandwidth of the system. Therefore, the routines that perform the multiplication and the main signal processing algorithms need to be optimized for speed in order to obtain the highest possible bandwidth when using the PIC16C74.

The selection of word size and coefficient scaling are also important factors in the successful implementation of signal processing systems. The effects of using a fixed word length to represent the signal and do calculations fall into three categories: signal quantization, round-off error, and coefficient quantization. The signal quantization due to the A/D converter and round-off error due to the finite precision arithmetic affect the overall signal-to-noise performance of the system. Scaling of the input signal should be done before the A/D converter to use the full input range and maximize the input signal-to-noise ratio. The use of double precision math for all calculations and storing intermediate results, even if the input and output signals are represented as 8-bit words, will help to reduce the round-off error noise to acceptable levels. Coefficient quantization occurs when the calculated coefficients are truncated or rounded off to fit within the given word length. This has the effect of moving the system transfer function poles and zeros which can change the system gain, critical frequencies of filters, or stability of the system. The successful implementation of these systems requires careful design and modeling of these effects using one of the many software programs that are available. The code written for this application note was first modeled using PC MATLAB before being implemented on the PIC16C74.

The algorithms in this application note are all implemented using fixed point two's compliment arithmetic. Two math libraries were used for the examples: one 8-bit signed multiply routine that was written specifically for the tone generation algorithm, and the modified double precision routines for the PIC16C54 that were used in the filtering routine. All numbers are stored in fractional two's compliment format where the MSB is the sign bit and there is an implied decimal point right after it. This is commonly referred to as Qx format where the number after the Q represents the number of fractional bits in the word. For instance, 16 bit words with the decimal point after the MSB would be referred to as Q15. This format allows numbers over the range of -1 to 0.99 to be represented and, because the magnitude of all numbers is less than or equal to one, has the advantage that there can be no overflow from a multiplication operation.

Since calculations are done using two's compliment arithmetic, values read by the PIC16C74's A/D converter need to be converted to this format. This can be easily done if the input is set up to read values in offset binary format. In this representation, the most negative input voltage is assigned to the number 0, zero volts is assigned the number 128, and the most positive voltage input is assigned 255. Since the PIC16C74 has a unipolar input A/D converter, a bipolar input signal must be scaled to be between 0 and 5V. One way to accomplish this is to use an op-amp scaling and offset circuit. The signal should be centered at 2.5V and have a peak to peak voltage swing of 4 to 4.5V. The offset binary number can be converted to two's compliment format by simply complimenting the MSB of the word. Once the signal processing calculations are completed, the number can be converted back to offset binary by complimenting the MSB before it is written to the PWM module. A similar level shifting circuit can be used at the PWM output to restore the DC level of the signal. Using this technique allows a wide range of analog input voltages to be handled by the PIC16C74.

© 1997 Microchip Technology Inc.

A/D and D/A Conversion

The PIC16C74's internal 8-bit A/D converter and PWM modules can be used to implement analog I/O for the system. The A/D converter along with an external anti-aliasing filter provides the analog input for the system. Depending on the input signal bandwidth and the sampling frequency, the filter can be a simple single pole RC filter or a multiple pole active filter. The PWM output along with an external output "smoothing" filter provides the D/A output for the system. This can be a simple RC filter if the PWM frequency is much higher (five to ten times) than the analog signal that is being output. Alternatively, an active filter can also be used at the PWM output . Since the use of the A/D and PWM modules is covered in detail in the data sheet for the part, they will not be covered here. In addition, since the PIC16C74's A/D converter is similar to the PIC16C71 and the PWM module is the same as the PIC16C74, the use of these is also covered in application notes AN546, AN538, and AN539.

Appendix A contains the listing for the C module "*ANALOGIO.C*" that has the functions that read the A/D converter input, initialize the PWM module, and write 8-bit values to the PWM module. The number format (offset binary or two's compliment) for the A/D and PWM values as well as the PWM resolution and mode are set using "`#define`" pragmas at the beginning of the module. The `get_sample()` function takes the A/D input multiplexor channel number as an argument and returns the measured input value. The `init_PWM()` function takes the PWM period register PR2 value as an argument. The `write_PWM()` function takes the output values for PWM module1 and 2 and writes them to the appropriate registers using the specified resolution. If the second argument to the function is 0, the registers for PWM module 2 are unaffected. The PWM resolution is always 8-bits but the mode used depends on the PWM frequency.

The A/D conversions need to be performed at the system sample rate which requires that some form of sample clock be generated internally or input from an external source. One way to generate this clock internally, in software with minimal effort, is to use the Timer2 interrupt. Since Timer2 is used to generate the PWM period, enabling the Timer2 interrupt and using the Timer2 postscaler can generate an interrupt at periods that are integer divisors of the PWM period. The ISR can set a software "sample flag" that is checked by the main routine. Once the sample flag is asserted by the ISR, the main routine can then clear it and perform the signal processing operation, output the next sample, and then wait for the sample flag to be asserted true again. Alternatively, a separate timer/counter or external clock input can be used for the system sample clock. The latter two methods have the advantage that the PWM frequency can be set independent of the sampling period. For best results, the PWM frequency should be set for at least five times the maximum frequency of the analog signal that is bring reproduced. The example programs illustrate the use of both of the methods for generating an internal sample clock.

Tone Generation

For systems that need to provide audible feedback or to provide DTMF signaling for telcom applications, the PIC16C74's PWM module can be used to generate these signals. One way to do this is to output samples of a sinusoidal waveform to the PWM module at the system sampling rate. This method is relatively simple but is limited to single tones and may require large amounts of memory depending on the number of samples used per cycle of the waveform and the number of tones that need to be generated. A more efficient method of generating both single and dual-tone signals is to use a difference equation method. This method uses a difference equation that is derived from the z-transform of a sinusoid as follows:

The z-transform of a sinusoid is

$$\frac{z^{-1}\sin\omega T}{1 - 2z^{-1}\cos\omega T + z^{-2}}$$

where the period $\omega = 2\pi f$ and T is the sampling period.

If this is interpreted as the transfer function $H(z) = Y(z)/X(z)$ then the difference equation can be found taking the inverse z-transform and applying the associated shift theorem as follows:

rearranging:

$$Y(z)(1 - 2z^{-1}\cos\omega T + z^{-2}) = X(z)(z^{-1}\sin\omega T)$$

$$Y(z) = z^{-1}X(z)\sin\omega T + z^{-1}Y(z)2\cos\omega T - z^{-2}Y(z)$$

taking the inverse z-transform:

$$Z^{-1}[Y(z)] = Z^{-1}[z^{-1}X(z)\sin\omega T + z^{-1}Y(z)2\cos\omega T - z^{-2}Y(z)]$$

$$y(n) = \sin\omega T\, x(n-1) + 2\cos\omega T\, y(n-1) - y(n-2)$$

If we let $a = \sin\omega T$ and $b = \cos\omega T$, the equation can be written as:

$$y(n) = a\, x(n-1) + 2b\, y(n-1) - y(n-2)$$

thus we have a difference equation with coefficients *a* and *b*. Note that only two coefficients are needed to generate a sinusoidal output sequence. These are calculated from the relationship above and stored in memory for use by the tone generation algorithm.

If we input an impulse to this system (x(n) = 1 at n = 0 and is zero elsewhere) then the output of the system will be a discrete-time sinusoidal sequence. Note that at n = 0, the output will always be 0 and x(n) is only 1 at n = 1 so the sequence becomes:

$$y(0) = 0$$
$$y(1) = a$$
$$y(n) = 2b\, y(n-1) - y(n-2)$$
for n equal to or greater than 2

© 1997 Microchip Technology Inc.

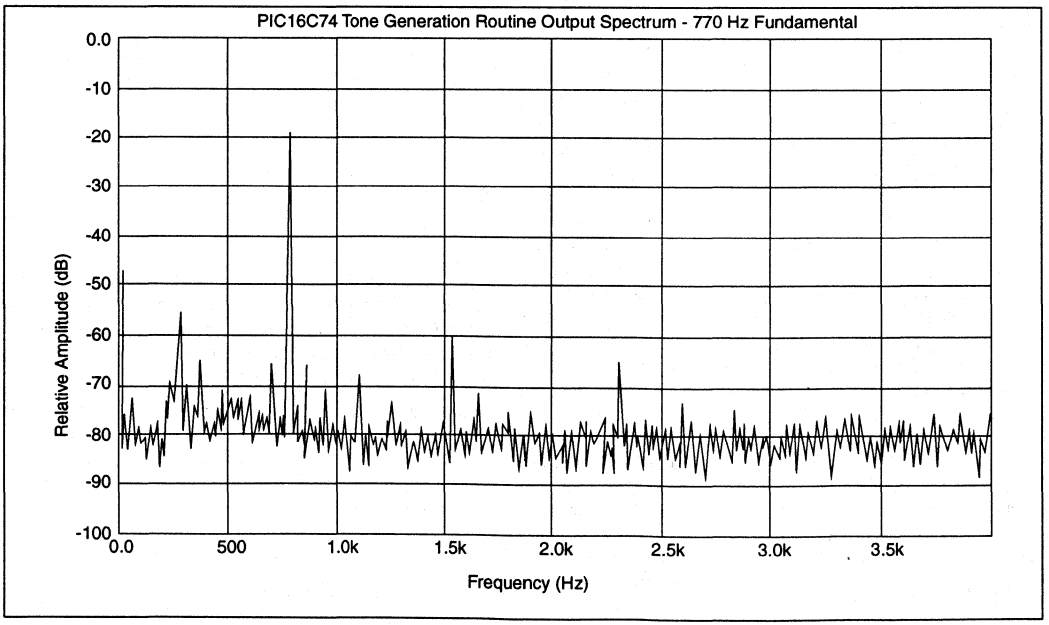
In order to further simplify the implementation of the algorithm, we can omit the first sample period. Since the output is already at 0 before starting, this will make no difference in the final output other than the fact that it will be time shifted by one sample. To generate dual tones, the algorithm is executed once for each tone and the two output samples are summed together. Since the output must be calculated and output to the D/A each sample period, a limitation exists on the frequency of the tone that can be produced for a given sample rate and processor speed. The higher the ratio of the sample clock to the tone frequency, the better, but a sample rate of at least three to four times the highest tone output should produce a sine wave with acceptable distortion.

Appendix B contains the listing for the "PICTONE.C" module which uses the difference equation method to produce variable length tones from the PWM module. Timer2 is used to generate the PWM period as well as the sample clock and tone duration timer. To send a tone, the coefficients and duration are written to the appropriate variables and then the tone routine is called. If the a2 and b2 coefficients are cleared, the routine will only generate a single tone sequence. The difference equation algorithm uses 8-bit signed math routines for the multiply operations. Using 8-bit coefficients reduces the accuracy by which the tones can be generated but greatly reduces the number of processor cycles needed to perform the algorithm since only single precision arithmetic is used. The spectrum of a single tone signal generated using this routine is shown in Figure 2.

Note that the second harmonic is better than 40 dB below the fundamental. Accuracy of this particular tone is better than 0.5%.

An example program "DTMFGEN.C" illustrates the use of the tone module to generate the 16 standard DTMF tones used for dialing on the telephone system. A sampling rate of 6.5 kHz was used which allows dual tones to be generated on a processor running at 10 MHz. Accuracy with respect to the standard DTMF frequencies is better than 1% for all tones and all harmonics above the fundamental frequency are greater than 30 dB down.

FIGURE 2: SINGLE TONE SIGNAL



© 1997 Microchip Technology Inc.

Digital Filters

Digital filters with critical frequencies up to a kilohertz or so can be implemented on the PIC16C74. Digital filters fall into two classes: Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters. FIR filters require more coefficients and multiplication operations to implement practical filters and are not as well suited for implementation on the PIC16C74. IIR type filters are typically designed starting with an analog filter prototype and then performing an analog to digital transformation to produce the digital filter coefficients. The subject of digital filter design is not within the scope of this application note but there are many excellent texts that cover the theory and design of these filters.

The implementation of a second-order IIR filter is done by using a second-order difference equation. A second-order infinite impulse response (IIR) filter has a transfer function of the form:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Where a_1 , a_2, b_0 , b_1 , and b_2 are the coefficients of the polynomials of the system transfer function that, when factored, yield the system poles and zeros. The difference equation found by taking the inverse z-transform and applying the shift theorem is:

$$y(n) =$$

$$b_0x(n) + b_1x(n - 1) + b_2x(n - 2) - a_1y(n - 1) - a_2y(n - 2)$$

Since the transfer function coefficients are used directly in the difference equation, this is often called the "Direct Form I" implementation of a digital filter. This form has its limitations due to numerical accuracy issues but is effective for implementing second-order systems.

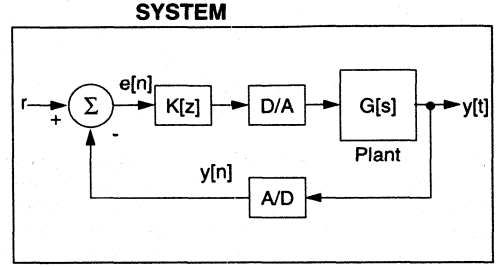
Appendix C contains the listing for the general-purpose filter routine "IIR_FILT.C" that can be used to implement low-pass, high-pass, bandpass, and bandstop (notch) filters. The filter() function takes an 8-bit input value x(n) and calculates the output value y(n) . The filter coefficients are stored as 16-bit two's compliment numbers and computation of the output is done using double precision arithmetic. Since the coefficients generated from the filter design program will be in decimal form, they need to be scaled to be less than 1 and then multiplied by 32,768 to put them in Q15 format. Additional scaling by factors of two may be required to prevent overflow of the sum during calculations. If this is done, the output must be multiplied by this scale factor to account for this. The "IIR_FILT.C" module contains two other subroutines required for the filtering program. One if these is a decimal adjust subroutine to restore the decimal place after two 16-bit Q15 numbers are multiplied. The subroutine shifts the 32-bit result left by one to get rid of the extra sign bit. The other routine scales the output by factors of two and is used after the output of the filter has been calculated to account for the scaling of the coefficients.

An example program "NOTCH_60.C" is provided that illustrates the implementation of a 60 Hz notch filter using the "IIR_FILT.C" module. The filter was modeled and designed using PC MATLAB before being implemented on the PIC16C74. A sample rate of 1 kHz is used which means that signals up to a few hundred hertz can be processed. The filter provides an attenuation of about 40 dB at 60 Hz and can be used to remove interference from sensor signals in a system.

Digital Control

A low bandwidth digital control system can be implemented on the PIC16C74 using the analog I/O and IIR filter routines. A typical digital control system is shown below:

FIGURE 3: TYPICAL DIGITAL CONTROL SYSTEM



The input, r, is the reference input and *y(t)* is the continuous-time output of the system. *G(s)* is the analog transfer function of the plant (controlled system) and *K(z)* is the digital compensator. The error signal is calculated by subtracting the measured output signal, *y(n)*, from the reference. The controller transfer function is essentially a filter that is implemented in the time-domain using a difference equation. Since digital control system design is a complex subject and the design of a suitable compensator depends on the system being controlled and the performance specifications, only the implementation issues will be discussed.

One popular and well understood compensator is the Proportional-Integral-Derivative (PID) controller whose transfer function is of the form:

$$K(z) = K_P + \frac{K_I}{1 - z^{-1}} + K_D(1 - z^{-1})$$

Where K_P is the proportional gain, K_I is the integral gain ,and K_D is the derivative gain. The transfer function can be implemented directly or can be put in the form of a standard second-order difference equation from the modified transfer function as shown below:

$$H(z) = \frac{(K_I T^2 + K_P T + K_D) - (2K_D + K_P T)z^{-1} + K_D z^{-2}}{T(1 - z^{-1})}$$

$$y(n) = (K_P + K_I T + \frac{K_D}{T})x(n) - (K_P + \frac{2K_D}{T})x(n - 1) + \frac{K_D}{T} x(n - 2) - y(n - 1)$$

Since the numerator coefficients will be greater than one, a gain factor K needs to be factored out so that the resulting coefficients are less than one. In this way, the IIR filter routine can be used to implement the controller. After the filter routine, the output y needs to be multiplied by K before being output to the PWM module. Since the gain can be high, this result needs to be checked for overflow and limited to the maximum 8-bit value, if required. Saturating the final result prevents the system from going unstable if overflow in the math does occur. The gains can also be applied externally at the D/A output. For example, the PWM can drive a power op-amp driver that provides a ± 20 volt swing for a DC motor.

RESULTS AND CONCLUSION

The results obtained using the PIC16C74 in these applications were impressive. The tone generation routines produce very clean sinusoidal signals and DTMF tones generated using this routine have been used to dial numbers over the telephone system with excellent results. In addition, tones used for audible feedback are more pleasing to the ear than those generated from a port pin as is typically done on processors without PWM modules. Using the PIC16C74 to generate these tones eliminates the need for special DTMF generator IC's thus reducing the cost and simplifying the design. The tone routine requires approximately 125 instruction cycles to calculate an output sample for a single tone output and 230 instruction cycles to calculate an output sample for a dual tone output.

The IIR filtering routines produce good results and have been used to filter 60 Hz signals on sensor lines and also to implement a simple PID controller system with excellent results. The IIR routine takes approximately 1670 instruction cycles to calculate the output. Table 1 shows the performance that can be expected with the PIC16C74 for various processor speeds.

In conclusion, the PIC16C74 provides the necessary performance to provide these simple, low bandwidth signal processing operations. This means that products using this device can benefit from cost and power savings by eliminating specialized components that normally perform these tasks.

References

Antoniou, A. Digital Filters: Analysis and Design. NY: McGraw-Hill Book Co., 1979.

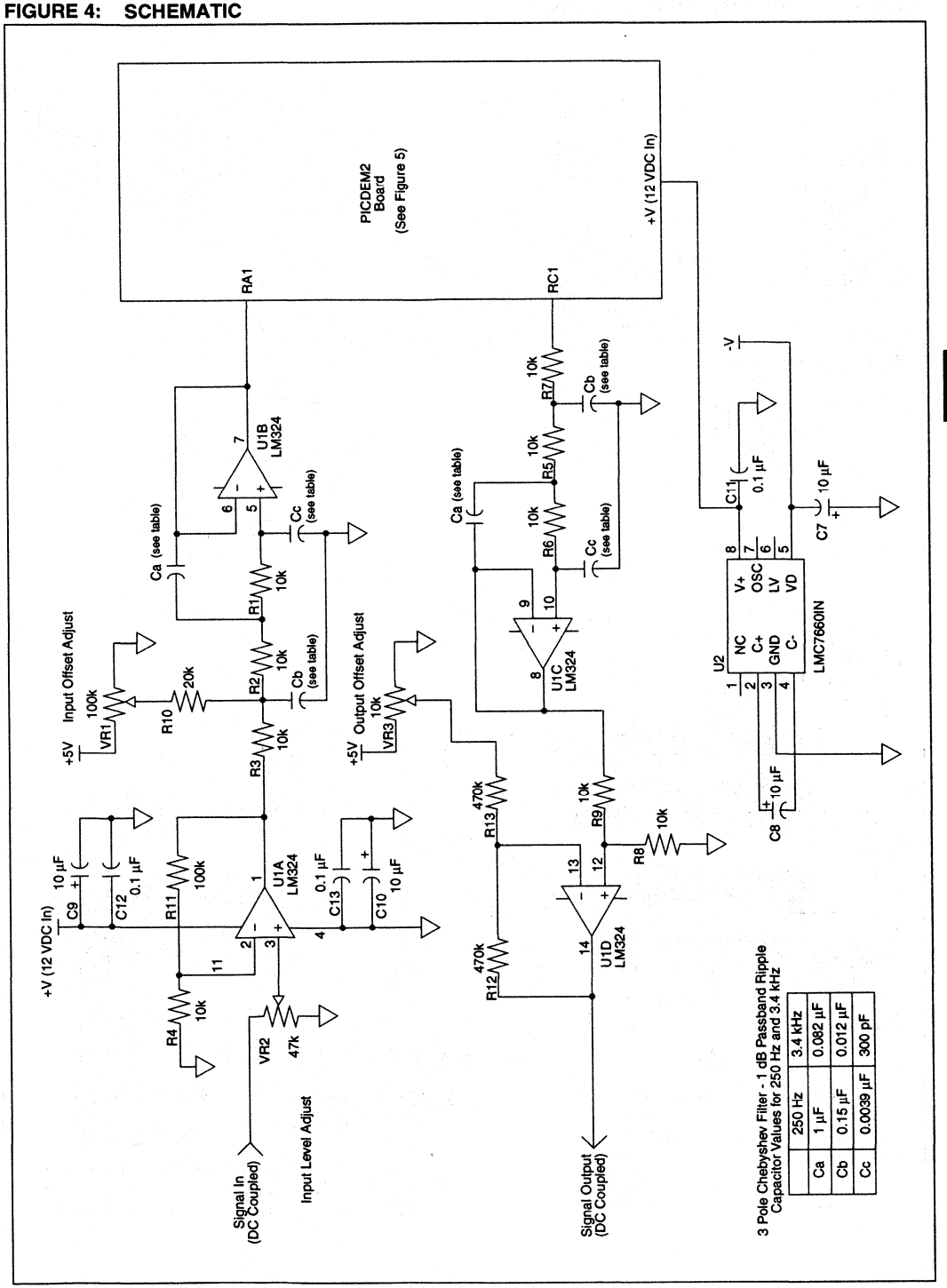
Openheim, A.V. and Schafer, R.W. Digital Signal Processing. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1975.

TABLE 1: PIC16C74 IIR FILTER PERFORMANCE

	4 MHz	8 MHz	10 MHz	16 MHz	20 MHz
A/D Input (35 cycles + 15 µs)	50 µs	32.5	29	23.75	22
IIR Filter (1850 cycles)	1850	925	740	462.5	370
PWM Output (62 cycles)	62	31	24.8	15.5	12.4
Total	1962	988.5	793.8	501.75	368.4
Max. Sampling Frequency	~500 Hz	~1000 Hz	~1250 Hz	~2000 Hz	~2500 Hz

© 1997 Microchip Technology Inc.

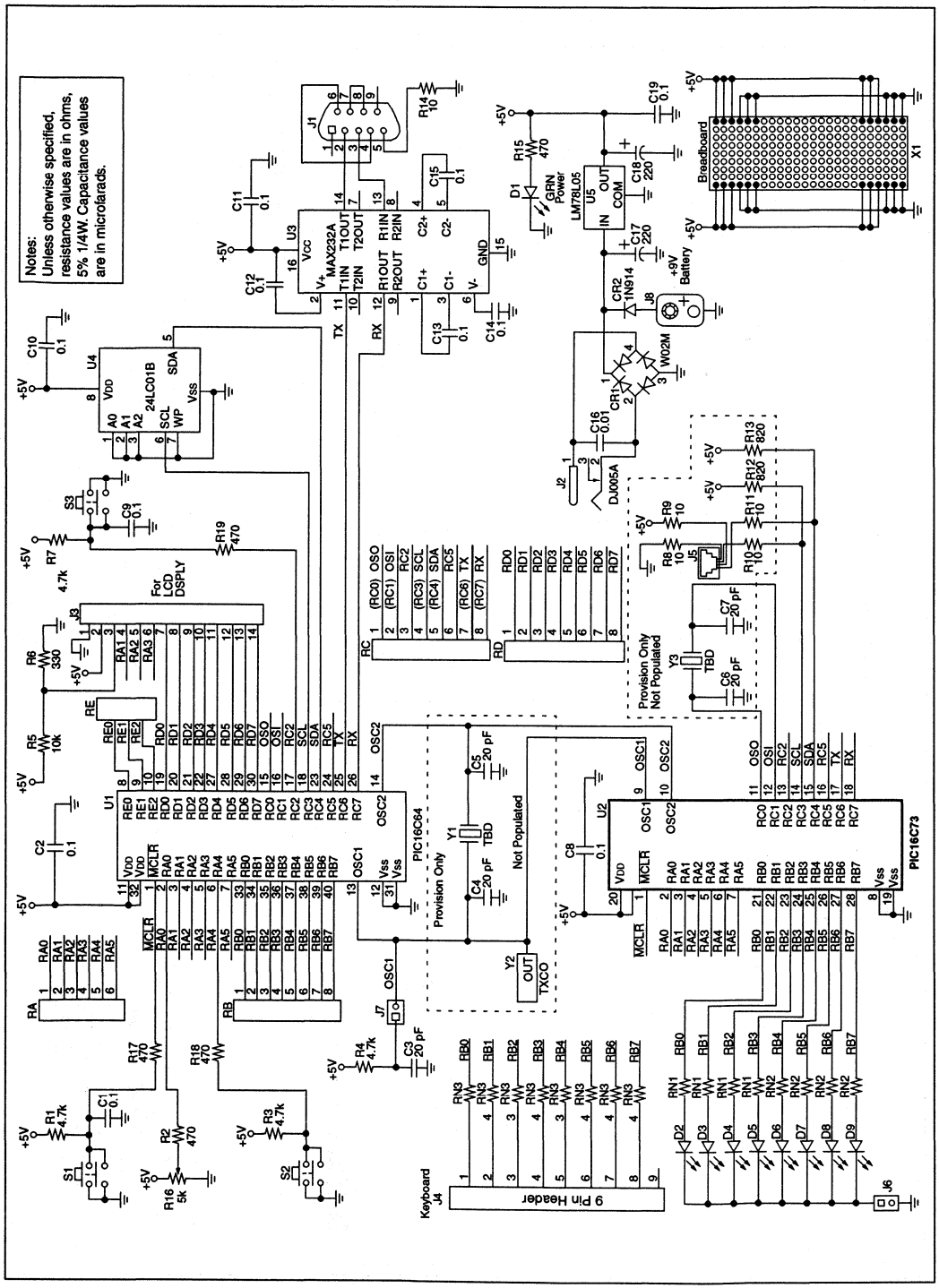
FIGURE 4: SCHEMATIC



3 Pole Chebyshev Filter - 1 dB Passband Ripple
Capacitor Values for 250 Hz and 3.4 kHz

	250 Hz	3.4 kHz
Ca	1 µF	0.082 µF
Cb	0.15 µF	0.012 µF
Cc	0.0039 µF	300 pF

FIGURE 5: PICDEM2 SCHEMATIC TIE-IN



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: ANALOG I/O MODULE

```
/************************************************************************
* Analog I/O Module
*
* Written for "Digital Signal Processing with the PIC16C74" Application Note
*
* This module contains functions that read the A-D inputs, initialize the PWM
* ports, and write values to the PWM ports.
*
* D. Mostowfi 4/95
************************************************************************/
#define active          1       /* define active as 1 */
#define LOW             0       /* define LOW as 0 */
#define HIGH            1       /* define HIGH as 1 */

#define OFFSET          0       /* define offset binary mode as 0 */
#define TWOS            1       /* define two's compliment mode as 1 */

#define AD_FORMAT       TWOS    /* define A-D format as TWOS */
#define PWM_FORMAT      TWOS    /* define PWM format as TWOS */
#define PWM_RES         HIGH    /* define PWM resolution as HIGH */

bits FLAGS;                     /* general purpose flags */
#define sample_flag     FLAGS.1 /* define sample_flag as FLAGS.1 */


/************************************************************************
* A-D Converter Routine - reads A-D converter inputs
*
* usage:
*       - call get_sample(channel #)
*       - returns 8 bit value
************************************************************************/
char get_sample(char channel)
{
char i;

    ADRES=0;                    /* clear ADRES */
    STATUS.C=0;                 /* clear carry */
    RLCF(channel);              /* and rotate channel 3 times */
    RLCF(channel);              /* to put in proper position */
    RLCF(channel);              /* for write to ADCON0 */
    ADCON0=channel;             /* write channel to ADCON0 */
    ADCON0.0=1;                 /* turn on A-D */
    i=0;                        /* set delay loop variable to 0 */
    while(i++<=5){};            /* delay (to ensure min sampling time) */
    ADCON0.2=1;                 /* start conversion */
    while(ADCON0.2){}           /* wait for eoc */
    ADCON0.0=0;                 /* turn off a-d converter */
    if(AD_FORMAT==TWOS){        /* if format is two's compliment */
      ADRES.7=!ADRES.7;         /* compliment MSB */
    }
    return ADRES;               /* return value in a-d result reg */
}


/************************************************************************
* PWM Initialization Routine - sets up PR2, sets output to mid-point, and
* starts timer 2 with interrupts disabled.
*
* usage:
```

```
*       - call init_PWM(PR2 register value)
*****************************************************************************/
void init_PWM(char _pr2)
{
    PR2=_pr2;                   /* reload value for 40khz PWM period */
    CCP1CON.5=0;                /* set CCPxCON = 0 for 50% output */
    CCP1CON.4=0;
    CCP2CON.5=0;
    CCP2CON.4=0;
    if(PWM_RES==HIGH){          /* if resolution is high, set CCPRxH=0 and */
      CCPR1H=0x00;              /* CCPRxL=0x20 for 50% PWM duty cycle */
      CCPR1L=0x20;
      CCPR2H=0x00;
      CCPR2L=0x20;
    }
    else{
      CCPR1H=0x00;              /* if resolution is low, set CCPRxH=0 and */
      CCPR1L=0x80;              /* CCPRxL=0x80 for 50% PWM duty cycle */
      CCPR2H=0x00;
      CCPR2L=0x80;
    }
    T2CON.TMR2ON=1;             /* start timer 2 */
    PIE1.TMR2IE=0;              /* and disable timer 2 interrupt */

}


/*****************************************************************************
* PWM Output Routine - writes output values to PWM ports
*
* Both high resolution and low resolution modes write 8 bit values - use of
* high or low resolution depends on PWM output period.
*
* usage:
*       - call write_PWM(channel 1 value, channel 2 value)
*         if channel 2 value=0, PWM port 2 not written
*****************************************************************************/
void write_PWM(bits pwm_out1, bits pwm_out2)
{

    if(PWM_FORMAT==TWOS){       /* if format is two's compliment */
      pwm_out1.7=!pwm_out1.7;   /* compliment msb's */
      pwm_out2.7=!pwm_out1.7;
    }
    if(PWM_RES==HIGH){          /* if resolution is high */
      STATUS.C=0;               /* clear carry */
      pwm_out1=RRCF(pwm_out1);  /* rotate right and write two lsb's */
      CCP1CON.4=STATUS.C;       /* to CCP1CON4 and CCP1CON5 */
      STATUS.C=0;
      pwm_out1=RRCF(pwm_out1);
      CCP1CON.5=STATUS.C;
      if(pwm_out2!=0){          /* if pwm_out2 not 0, do the same */
        STATUS.C=0;             /* for channel 2 */
        pwm_out2=RRCF(pwm_out2);
        CCP2CON.4=STATUS.C;
        STATUS.C=0;
        pwm_out2=RRCF(pwm_out2);
        CCP2CON.5=STATUS.C;
      }
    }
    CCPR1L=pwm_out1;            /* write value to CCPR1L */
    if(pwm_out2!=0){            /* if pwm_out2 not 0, do the same */
      CCPR2L=pwm_out2;          /* for CCPR2L */
    }
}                               /* done */
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: TONE GENERATION MODULE

```
/************************************************************************
* Tone Generation Module
*
* Written for "Digital Signal Processing with the PIC16C74" Application Note.
*
* This module contains a C callable module that generates single or dual
* tones using a difference equation method:
*
* y1(n)=a1*x(n-1)+b1*y1(n-1)-y1(n-2)
* y2(n)=a2*x(n-1)+b2*y2(n-1)-y2(n-2)
*
* The routine is written in assembly language and uses the optimized signed
* 8x8 multiply routine and scaling routine in the file 8BITMATH.C.
*
* D. Mostowfi 2/95
************************************************************************/
#include "\mpc\apnotes\8bitmath.c"  /* 8 bit signed math routines */

#define sample_flag FLAGS.1         /* sample flag */
#define no_tone2 FLAGS,2            /* no tone 2 flag */

extern char ms_cntr;                /* millisecond counter for tone loop */

char a1;                            /* first tone (low-group) coeeficient 1 */
char a2;                            /* first tone (low-group) coefficient 2 */
char b1;                            /* second tone (high group) coefficient 1 */
char b2;                            /* second tone (high group) coefficient 2 */
char duration;                      /* tone duration */

char y1;                            /* output sample y1(n) for tone 1 */
char y2;                            /* output sample y2(n) for tone 2 */


/************************************************************************
* Tone function - generates single or dual tone signals out PWM port 1.
*
* usage:
*       - write coefficients for tone 1 to a1 and b1
*       - write coefficents for tone 2 to a2 and b2 (0 if no tone 2)
*       - write duration of tone in milliseconds to duration
*       - call tone() function
************************************************************************/
void tone(void)
{

char y1_1;                          /* y1(n-1) */
char y1_2;                          /* y1(n-2) */
char y2_1;                          /* y2(n-1) */
char y2_2;                          /* y2(n-2) */

    PIR1.TMR2IF=0;                  /* clear timer 2 interrupt flag */
    PIE1.TMR2IE=1;                  /* and enable timer 2 interrupt */
    ms_cntr=0;                      /* clear ms counter */
    STATUS.RP0=0;                   /* set proper bank!!! */

#asm
        clrf    y1                  ; clear output byte and taps
        clrf    y2                  ;
        clrf    y1_1                ;
        clrf    y1_2                ;
```

```
        clrf    y2_1            ;
        clrf    y2_2            ;

        bcf     no_tone2        ; clear no tone 2 flag
        clrf    ms_cntr         ; clear millisecond counter

first_sample:
        movf    a1,W            ; first iteration
        movwf   y1              ; y1(n)=a1
        movwf   y1_1            ;
        movlw   0x00            ;
        iorwf   a2,W            ;
        btfsc   STATUS,Z        ; generate second tone (a2 !=0) ?
        bsf     no_tone2        ;
        movf    a2,W            ; y2(n)=a2
        movwf   y2              ;
        movwf   y2_1            ;
        movf    y2,W            ;
        addwf   y1,F            ; y1(n)=y1(n)+y2(n) (sum two tone outputs)

tone_loop:
        movf    ms_cntr,W       ; test to see if ms=duration (done?)
        subwf   duration,W      ;
        btfsc   STATUS,Z        ;
        goto    tone_done       ;

wait_PWM:
        btfss   FLAGS,1         ; test sample flag (sample period elapsed?)
        goto    wait_PWM        ; loop if not

        bcf     FLAGS,1         ; if set, clear sample flag

#endasm
    write_PWM((char)y1,0);      /* write y1 to PWM port */
#asm

next_sample:
        movf    b1,W            ; y1(n)=b1*y1(n-1)-y1(n-2)
        movwf   multcnd         ;
        movf    y1_1,W          ;
        movwf   multplr         ;
        call    _8x8smul        ;
        call    scale_16        ;
        movf    y1_2,W          ;
        subwf   result_1,W      ;
        movwf   y1              ;
        movf    y1_1,W          ; y1(n-2)=y1(n-1)
        movwf   y1_2            ;
        movf    y1,W            ; y1(n-1)=y1(n)
        movwf   y1_1            ;
        btfsc   no_tone2        ;
        goto    tone_loop       ;
        movf    b2,W            ; y2(n)=b2*y2(n-1)-y2(n-2)
        movwf   multcnd         ;
        movf    y2_1,W          ;
        movwf   multplr         ;
        call    _8x8smul        ;
        call    scale_16        ;
        movf    y2_2,W          ;
        subwf   result_1,W      ;
        movwf   y2              ;
        movf    y2_1,W          ; y2(n-2)=y2(n-1)
        movwf   y2_2            ;
        movf    y2,W            ; y2(n-1)=y2(n)
        movwf   y2_1            ;
```


© 1997 Microchip Technology Inc.

```
        movf    y2,W            ;
        addwf   y1,F            ; y1(n)=y1(n)+y2(n) (sum two tone outputs)

        goto    tone_loop       ; go and calculate next sample

tone_done:

#endasm

    CCP1CON.5=0;                /* reset PWM outputs to mid value */
    CCP1CON.4=0;
    CCP2CON.5=0;
    CCP2CON.4=0;
    CCPR1H=0x00;
    CCPR1L=0x20;
    CCPR2H=0x00;
    CCPR2L=0x20;

    PIE1.TMR2IE=0;              /* disable timer 2 interrupts */
    PIR1.TMR2IF=0;              /* and clear timer 2 interrupt flag */
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C: DTMF TONE GENERATION

```
/************************************************************************
* DTMF tone generation using PIC16C74
*
* Written for the "Digital Signal Processing Using the PIC16C74" Ap Note
*
* Generates 16 DTMF tones (1-9,0,*,#,A,B,C,D) out PWM port 1
*
* Uses PICTONE.C and ANALOGIO.C modules
*
* D. Mostowfi 4/95
************************************************************************/
#include "\mpc\include\delay14.h"
#include "\mpc\include\16c74.h"      /* c74 header file */
#include "\mpc\math.h"


#include "\mpc\apnotes\analogio.c"   /* analog I/O module */
#include "\mpc\apnotes\pictone.c"    /* tone generation module */

bits pwm1;

/* Function Prototypes */

void main_isr();
void timer2_isr();

/* 16C74 I/O port bit declarations */


/* global program variables */

char tmr2_cntr;                     /* timer 2 interrupt counter */
char delay_cntr;                    /* delay time counter (10ms ticks)*/


/* Tone Coefficients for DTMF Tones */

const DTMF_1[4]={30, 51, 48, 27};
const DTMF_2[4]={30, 51, 56, 19};
const DTMF_3[4]={30, 51, 64, 11};
const DTMF_4[4]={33, 48, 48, 27};
const DTMF_5[4]={33, 48, 56, 19};
const DTMF_6[4]={33, 48, 64, 11};
const DTMF_7[4]={36, 45, 48, 27};
const DTMF_8[4]={36, 45, 56, 19};
const DTMF_9[4]={36, 45, 64, 11};
const DTMF_0[4]={40, 41, 56, 19};
const DTMF_star[4]={40, 41, 48, 27};
const DTMF_pound[4]={40, 41, 64, 11};
const DTMF_A[4]={30, 51, 75, 2};
const DTMF_B[4]={33, 48, 75, 2};
const DTMF_C[4]={36, 45, 75, 2};
const DTMF_D[4]={40, 41, 75, 2};


/************************************************************************
* main isr - 16C74 vectors to 0004h (MPC __INT() function) on any interrupt *
* assembly language routine saves W and Status registers then tests flags in
* INTCON to determine source of interrupt. Routine then calls appropriate isr.
* Restores W and status registers when done.
```


© 1997 Microchip Technology Inc.

```
*************************************************************************/
void __INT(void)
{

  if(PIR1.TMR2IF){              /* timer 2 interrupt ? */
    PIR1.TMR2IF=0;              /* clear interrupt flag */
    timer2_isr();               /* and call timer 2 isr */
  }

/* Restore W, WImage, and STATUS registers */

#asm
    BCF     STATUS,RP0              ;Bank 0
    MOVF    temp_PCLATH, W
    MOVWF   PCLATH                  ;PCLATH restored
    MOVF    temp_WImage, W
    MOVWF   __WImage                ;__WImage restored
    MOVF    temp_FSR, W
    MOVWF   FSR                     ;FSR restored
    SWAPF   temp_STATUS,W
    MOVWF   STATUS                  ;STATUS restored
    SWAPF   temp_WREG,F
    SWAPF   temp_WREG,W             ;W restored
#endasm

}

/*************************************************************************
* timer 2 isr - provides PWM sample clock generation and millisecond counter
* for tone routine
*************************************************************************/
void timer2_isr(void)
{
     sample_flag=active;        /* set sample flag (150us clock) */
     PORTB.7=!PORTB.7;          /* toggle PORTB.7 at sample rate */
     if(tmr2_cntr++==7){        /* check counter */
       tmr2_cntr=0;             /* reset if max */
       ms_cntr++;               /* and increment millisecond ticks */
     }
}


void main()
{

/* initialize OPTION register */
   OPTION=0b11001111;

/* initialize INTCON register (keep GIE inactive!) */
   INTCON=0b00000000;           /* disable all interrupts */

/* initialize PIE1 and PIE2 registers (peripheral interrupts) */
   PIE1=0b00000000;             /* disable all interrupts */
   PIE2=0b00000000;

/* initialize T1CON and T2CON registers */
   T1CON=0b00000000;            /* T1 not used  */
   T2CON=0b00101000;            /* T2 postscaler=5  */

/* initialize CCPxCON registers */
   CCP1CON=0b00001100;          /* set CCP1CON for PWM mode */
   CCP2CON=0b00001100;          /* set CCP2CON for PWM mode (not used in demo) */

/* initialize SSPCON register */
   SSPCON=0b00000000;           /* serial port - not used */
```


© 1997 Microchip Technology Inc.

```
/* initialize ADCONx registers */
   ADCON0=0b00000000;        /* A-D converter */
   ADCON1=0b00000010;

/* initialize TRISx register (port pins as inputs or outputs) */
   TRISA=0b00001111;
   TRISB=0b00000000;
   TRISC=0b10000000;
   TRISD=0b00001111;
   TRISE=0b00000000;

/* clear watchdog timer (not used) */
   CLRWDT();

/* initialize program variables */
   tmr2_cntr=0;

/* initialize program bit variables */
   FLAGS=0b00000000;

/* intialize output port pins (display LED's on demo board) */
   PORTB=0;

/* enable interrupts...  */

   INTCON.ADIE=1;            /* Peripheral interrupt enable */
   INTCON.GIE=1;             /* global interrupt enable */



   init_PWM(0x3e);           /* initialize PWM port */

   PORTB=0x01;               /* write a 1 to PORTB */
   a1=DTMF_1[0];             /* and send a DTMF "1" */
   b1=DTMF_1[1];
   a2=DTMF_1[2];
   b2=DTMF_1[3];
   duration=150;
   tone();
   Delay_Ms_20MHz(200);      /* delay 100ms (200/2 using MPC delays) */


   PORTB=0x02;               /* write a 2 to PORT B */
   a1=DTMF_2[0];             /* and send a DTMF "2" */
   b1=DTMF_2[1];
   a2=DTMF_2[2];
   b2=DTMF_2[3];
   duration=150;
   tone();
   Delay_Ms_20MHz(200);      /* delay 100ms (200/2 using MPC delays) */


   PORTB=0x03;               /* write a 3 to PORTB */
   a1=DTMF_3[0];             /* and send a DTMF "3" */
   b1=DTMF_3[1];
   a2=DTMF_3[2];
   b2=DTMF_3[3];
   duration=150;
   tone();
   Delay_Ms_20MHz(200);      /* delay 100ms (200/2 using MPC delays) */


   PORTB=0x04;               /* write a 4 to PORTB */
   a1=DTMF_4[0];             /* and send a DTMF "4" */
   b1=DTMF_4[1];
```


© 1997 Microchip Technology Inc.

```
a2=DTMF_4[2];
b2=DTMF_4[3];
duration=150;
tone();
Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */
PORTB=0x05;           /* write a 5 to PORTB */
a1=DTMF_5[0];         /* and send a DTMF "5" */
b1=DTMF_5[1];
a2=DTMF_5[2];
b2=DTMF_5[3];
duration=150;
tone();
Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */


PORTB=0x06;           /* write a 6 to PORTB */
a1=DTMF_6[0];         /* and send a DTMF "6" */
b1=DTMF_6[1];
a2=DTMF_6[2];
b2=DTMF_6[3];
duration=150;
tone();
Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */


PORTB=0x07;           /* write a 7 to PORTB */
a1=DTMF_7[0];         /* and send a DTMF "7" */
b1=DTMF_7[1];
a2=DTMF_7[2];
b2=DTMF_7[3];
duration=150;
tone();
Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */


PORTB=0x08;           /* write a 8 to PORTB */
a1=DTMF_8[0];         /* and send a DTMF "8" */
b1=DTMF_8[1];
a2=DTMF_8[2];
b2=DTMF_8[3];
duration=150;
tone();
Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */


PORTB=0x09;           /* write a 9 to PORTB */
a1=DTMF_9[0];         /* and send a DTMF "9" */
b1=DTMF_9[1];
a2=DTMF_9[2];
b2=DTMF_9[3];
duration=150;
tone();
Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */


PORTB=0x0;            /* write a 0 to PORTB */
a1=DTMF_0[0];         /* and send a DTMF "0" */
b1=DTMF_0[1];
a2=DTMF_0[2];
b2=DTMF_0[3];
duration=150;
tone();
Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */
Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */
```


© 1997 Microchip Technology Inc.

```
    PORTB=0x0e;             /* write a 0x0e to PORTB */
    a1=DTMF_star[0];        /* and send a DTMF "*" */
    b1=DTMF_star[1];
    a2=DTMF_star[2];
    b2=DTMF_star[3];
    duration=250;
    tone();
    Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */


    PORTB=0x0f;             /* write a 0x0f to PORTB */
    a1=DTMF_pound[0];       /* and send a DTMF "#" */
    b1=DTMF_pound[1];
    a2=DTMF_pound[2];
    b2=DTMF_pound[3];
    duration=250;
    tone();
    Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */
    Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */



    PORTB=0x0a;             /* write a 0x0a to PORTB */
    a1=DTMF_A[0];           /* and send a DTMF "A" */
    b1=DTMF_A[1];
    a2=DTMF_A[2];
    b2=DTMF_A[3];
    duration=250;
    tone();
    Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */


    PORTB=0x0b;             /* write a 0x0b to PORTB */
    a1=DTMF_B[0];           /* and send a DTMF "B" */
    b1=DTMF_B[1];
    a2=DTMF_B[2];
    b2=DTMF_B[3];
    duration=250;
    tone();
    Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */


    PORTB=0x0c;             /* write a 0x0c to PORTB */
    a1=DTMF_C[0];           /* and send a DTMF "C" */
    b1=DTMF_C[1];
    a2=DTMF_C[2];
    b2=DTMF_C[3];
    duration=250;
    tone();
    Delay_Ms_20MHz(200);  /* delay 100ms (200/2 using MPC delays) */



    PORTB=0x0d;             /* write a 0x0d to PORTB */
    a1=DTMF_D[0];           /* and send a DTMF "D" */
    b1=DTMF_D[1];
    a2=DTMF_D[2];
    b2=DTMF_D[3];
    duration=250;
    tone();

    PORTB=0;                /* write a 0 to PORTB */

    while(1){}              /* done (loop) */

}
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX D: IIR FILTER MODULE

```
/************************************************************************
* Second-Order IIR Filter Module
*
* Written for "Digital Signal Processing with the PIC16C74" Application Note.
*
* This routine implements an IIR filter using a second order difference
* equation of the form:
*
* y(n) = b0*x(n)+b1*x(n-1)+b2*x(n-2)+a1*y(n-1)+a2*y(n-2)
*
* D. Mostowfi 3/95
************************************************************************/
#include "\mpc\apnotes\dbl_math.c"

bits            x_n;            /* input sample x(n) */
unsigned long   y_n;            /* output sample y(n) */
unsigned long   x_n_1;          /* x(n-1) */
unsigned long   x_n_2;          /* x(n-2) */
unsigned long   y_n_1;          /* y(n-1) */
unsigned long   y_n_2;          /* y(n-2) */

char            rmndr_h;        /* high byte of remainder from multiplies */
char            rmndr_l;        /* low byte of remainder from multiplies */

#define A1_H    0xd2            /* filter coefficients */
#define A1_L    0x08            /* for 60Hz notch filter */
#define A2_H    0x11            /* Fs= 1kHz */
#define A2_L    0x71
#define B0_H    0x18
#define B0_L    0xbb
#define B1_H    0xd2
#define B1_L    0x08
#define B2_H    0x18
#define B2_L    0xb9


/************************************************************************
* Filter initialization - clears all taps in memory.
*
* usage:
*       - call init_filter()
*         use at program initialization
************************************************************************/
void init_filter(){

#asm

        clrf    y_n             ; clear output value
        clrf    y_n+1           ;
        clrf    y_n_1           ; and all filter "taps"
        clrf    y_n_1+1         ;
        clrf    y_n_2           ;
        clrf    y_n_2+1         ;
        clrf    x_n_1           ;
        clrf    x_n_1+1         ;
        clrf    x_n_2           ;
        clrf    x_n_2+1         ;

#endasm
```

```
}


/*****************************************************************************
* Assembly language subroutines for main filter() function
*****************************************************************************/
#asm

;
; Add Remainder subroutine - adds remainder from multiplies to ACCc
;

add_rmndr:
        btfss     sign,7          ; check if number is negative
        goto      add_r_start     ; go to add_r_start if not
        comf      ACCcLO          ; if so, negate number in ACC
        incf      ACCcLO          ;
        btfsc     STATUS,Z        ;
        decf      ACCcHI          ;
        comf      ACCcHI          ;
        btfsc     STATUS,Z        ;
        comf      ACCbLO          ;
        incf      ACCbLO          ;
        btfsc     STATUS,Z        ;
        decf      ACCbHI          ;
        comf      ACCbHI          ;

add_r_start:
        movf      rmndr_l,W       ;  get low byte of remainder
        addwf     ACCcLO          ;  and add to ACCcLO
        btfsc     STATUS,C        ;  check for overflow
        incf      ACCcHI          ;  if overflow, increment ACCcHI
        movf      rmndr_h,W       ;  get high byte of remainder
        addwf     ACCcHI          ;  and add to ACCcHI
        btfsc     STATUS,C        ;  check for overflow
        incf      ACCbLO          ;  if overflow, increment ACCbLO

        btfss     sign,7          ; check if result negative
        goto      add_r_done      ; if not, go to add_r_done
        comf      ACCcLO          ; if so, negate result
        incf      ACCcLO          ;
        btfsc     STATUS,Z        ;
        decf      ACCcHI          ;
        comf      ACCcHI          ;
        btfsc     STATUS,Z        ;
        comf      ACCbLO          ;
        incf      ACCbLO          ;
        btfsc     STATUS,Z        ;
        decf      ACCbHI          ;
        comf      ACCbHI          ;

add_r_done:
        retlw     0               ; done

;
; Decimal Adjust Subroutine - used after each Q15 multiply to convert Q30 result
; to Q15 number

dec_adjust:
        bcf       sign,7          ; clear sign
        btfss     ACCbHI,7        ; test if number is negative
        goto      adjust          ; go to adjust if not
        bsf       sign,7          ; set sign if negative

        comf      ACCcLO          ; and negate number
        incf      ACCcLO
```


© 1997 Microchip Technology Inc.

```
        btfsc   STATUS,Z
        decf    ACCcHI
        comf    ACCcHI
        btfsc   STATUS,Z
        comf    ACCbLO
        incf    ACCbLO
        btfsc   STATUS,Z
        decf    ACCbHI
        comf    ACCbHI

adjust:
        rlf     ACCcHI          ; rotate ACC left 1 bit
        rlf     ACCbLO          ;
        rlf     ACCbHI          ;

        btfss   sign,7          ; check if result should be negative
        goto    adj_done        ; if not, done
        comf    ACCbLO          ; if result negative, negate ACC
        incf    ACCbLO
        btfsc   STATUS,Z
        decf    ACCbHI
        comf    ACCbHI

adj_done:
        retlw   0               ; done

;
; Output Scaling Routine - used to scale output samples by factors of
; 2, 4, or 8 at end of filter routine
;
scale_y_n:
        bcf     sign,7          ; clear sign,7
        btfss   y_n+1,7         ; test if y(n) negative
        goto    start_scale     ; go to start_scale if not
        bsf     sign,7          ; set sign,7 if negative
        comf    y_n             ; and compliment y(n)
        incf    y_n             ;
        btfsc   STATUS,Z        ;
        decf    y_n+1           ;
        comf    y_n+1           ;

start_scale:
        bcf     STATUS,C        ; clear carry
        rlf     y_n+1           ; and rotate y(n) left
        rlf     y_n             ;
        bcf     STATUS,C        ;
        rlf     y_n+1           ;
        rlf     y_n             ;
        bcf     STATUS,C        ;
        rlf     y_n+1           ;
        rlf     y_n             ;

        btfss   sign,7          ; test if result is negative
        goto    scale_y_done    ; go to scale_y_done if not
        comf    y_n             ; negate y(n) if result is negative
        incf    y_n             ;
        btfsc   STATUS,Z        ;
        decf    y_n+1           ;
        comf    y_n+1           ;

scale_y_done:
        retlw   0               ; done

#endasm
```

```
/*****************************************************************************
* Filter function - filter takes current input sample, x(n), and outputs next
* output sample, y(n).
*
* usage:
*       - write sample to be filtered to x_n
*       - call filter()
*       - output is in MSB of y_n (y_n=MSB, y_n+1=LSB)
*
*****************************************************************************/
void filter(){

#asm

        clrf    y_n             ; clear y(n) before starting
        clrf    y_n+1           ;

        clrf    ACCbLO          ; move x(n) to ACCbHI
        movf    x_n,W           ; (scale 8 bit - 16 bit input)
        movwf   ACCbHI          ;

        movlw   B0_H            ; get coefficient b0
        movwf   ACCaHI          ; y(n)=b0*x(n)
        movlw   B0_L            ;
        movwf   ACCaLO          ;
        call    D_mpyF          ;
        movf    ACCcHI,W        ; save remainder from multiply
        movwf   rmndr_h         ;
        movf    ACCcLO,W        ;
        movwf   rmndr_l         ;
        call    dec_adjust      ;
        movf    ACCbHI,W        ;
        movwf   y_n+1           ;
        movf    ACCbLO,W        ;
        movwf   y_n             ;

        movlw   B1_H            ; get coefficient b1
        movwf   ACCaHI          ; y(n)=y(n)+b1*x(n-1)
        movlw   B1_L            ;
        movwf   ACCaLO          ;
        movf    x_n_1+1,W       ;
        movwf   ACCbHI          ;
        movf    x_n_1,W         ;
        movwf   ACCbLO          ;
        call    D_mpyF          ;
        call    add_rmndr       ; add in remainder from previous multiply
        movf    ACCcHI,W        ; and save new remainder
        movwf   rmndr_h         ;
        movf    ACCcLO,W        ;
        movwf   rmndr_l         ;
        call    dec_adjust      ;
        movf    y_n+1,W         ;
        movwf   ACCaHI          ;
        movf    y_n,W           ;
        movwf   ACCaLO          ;
        call    D_add           ;
        movf    ACCbHI,W        ;
        movwf   y_n+1           ;
        movf    ACCbLO,W        ;
        movwf   y_n             ;

        movlw   B2_H            ; get coefficient b2
        movwf   ACCaHI          ; y(n)=y(n)+b2*x(n-2)
        movlw   B2_L            ;
        movwf   ACCaLO          ;
        movf    x_n_2+1,W       ;
```


© 1997 Microchip Technology Inc.

```
movwf   ACCbHI          ;
movf    x_n_2,W         ;
movwf   ACCbLO          ;
call    D_mpyF          ;
call    add_rmndr       ; add in remainder from previous multiply
movf    ACCcHI,W        ; and save new remainder
movwf   rmndr_h         ;
movf    ACCcLO,W        ;
movwf   rmndr_l         ;
call    dec_adjust      ;
movf    y_n+1,W         ;
movwf   ACCaHI          ;
movf    y_n,W           ;
movwf   ACCaLO          ;
call    D_add           ;
movf    ACCbHI,W        ;
movwf   y_n+1           ;
movf    ACCbLO,W        ;
movwf   y_n             ;

movlw   A1_H            ; get coefficient a1
movwf   ACCaHI          ; y(n)=y(n)+a1*y(n-1)
movlw   A1_L            ;
movwf   ACCaLO          ;
movf    y_n_1+1,W       ;
movwf   ACCbHI          ;
movf    y_n_1,W         ;
movwf   ACCbLO          ;
call    D_mpyF          ;
call    add_rmndr       ; add in remainder from previous multiply
movf    ACCcHI,W        ; and save new remainder
movwf   rmndr_h         ;
movf    ACCcLO,W        ;
movwf   rmndr_l         ;
call    dec_adjust      ;
movf    y_n+1,W         ;
movwf   ACCaHI          ;
movf    y_n,W           ;
movwf   ACCaLO          ;
call    D_sub           ;
movf    ACCbHI,W        ;
movwf   y_n+1           ;
movf    ACCbLO,W        ;
movwf   y_n             ;

movlw   A2_H            ; get coefficient a2
movwf   ACCaHI          ; y(n)=y(n)+a2*y(n-2)
movlw   A2_L            ;
movwf   ACCaLO          ;
movf    y_n_2+1,W       ;
movwf   ACCbHI          ;
movf    y_n_2,W         ;
movwf   ACCbLO          ;
call    D_mpyF          ;
call    add_rmndr       ;
call    dec_adjust      ;
movf    y_n+1,W         ;
movwf   ACCaHI          ;
movf    y_n,W           ;
movwf   ACCaLO          ;
call    D_sub           ;
movf    ACCbHI,W        ;
movwf   y_n+1           ;
movf    ACCbLO,W        ;
movwf   y_n             ;
```


© 1997 Microchip Technology Inc.

```
        movf    x_n_1,W         ; x(n-2)=x(n-1)
        movwf   x_n_2           ;
        movf    x_n_1+1,W       ;
        movwf   x_n_2+1         ;
        movf    x_n,W           ; x(n-1)=x(n)
        movwf   x_n_1+1         ;
        clrf    x_n_1           ;

        movf    y_n_1,W         ; y(n-2)=y(n-1)
        movwf   y_n_2           ;
        movf    y_n_1+1,W       ;
        movwf   y_n_2+1         ;
        movf    y_n,W           ; y(n-1)=y(n)
        movwf   y_n_1           ;
        movf    y_n+1,W         ;
        movwf   y_n_1+1         ;

        call    scale_y_n       ;

        movf    y_n+1,W         ; shift lsb of y_n to msb
        movwf   y_n             ;

#endasm
}
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX E: NOTCH FILTER

```
/*************************************************************************
* 60 Hertz Notch Filter
*
* Written for "Digital Signal Processing with the PIC16C74" Application Note.
*
* This example program use the filter() function to implement a 60Hz notch
* filter. T0 is used to generate a 1kHz sample clock. The program samples the
* input signal x(n) on A-D channel 1, calls the filter routine  signal, and
* outputs y(n) to PWM channel 1.
*
* If FILTER set to 0, performs straight talkthru from A-D to PWM output.
* T0 period can be changed to cary the sample rate.
*
* D. Mostowfi 4/95
*************************************************************************/
#include "\mpc\include\16c74.h"          /* c74 header file */

#include "\mpc\apnotes\analogio.c"       /* analog I/O module */
#include "\mpc\apnotes\iir_filt.c"       /* iir filter module */

#define FILTER          1

/* Function Prototypes */
void main_isr();
void timer0_isr();


/*************************************************************************
* main isr - 16C74 vectors to 0004h (MPC __INT() function) on any interrupt *
* assembly language routine saves W and Status registers then tests flags in
* INTCON to determine source of interrupt. Routine then calls appropriate isr.
* Restores W and status registers when done.
*************************************************************************/
void __INT(void)
{

    if(INTCON.T0IF){                    /* timer 0 interrupt ? */
      INTCON.T0IF=0;                    /* clear interrupt flag */
      timer0_isr();                     /* and call timer 0 isr */
    }

/* Restore W, WImage, and STATUS registers */

#asm
        BCF     STATUS,RP0              ;Bank 0
        MOVF    temp_PCLATH, W
        MOVWF   PCLATH                  ;PCLATH restored
        MOVF    temp_WImage, W
        MOVWF   __WImage                ;WImage restored
        MOVF    temp_FSR, W
        MOVWF   FSR                     ;FSR restored
        SWAPF   temp_STATUS,W
        MOVWF   STATUS                  ;RP0 restored
        SWAPF   temp_WREG,F
        SWAPF   temp_WREG,W             ;W restored
#endasm

}

/*************************************************************************
* timer 0 interrupt service routine
*************************************************************************/
void timer0_isr(void)
```

```
{
    TMR0=100;                   /* reload value for 1ms period */
    PORTB.0=!PORTB.0;           /* toggle PORTB.0 */
    sample_flag=active;         /* set sample flag */
}


void main()
{

/* initialize OPTION register */
    OPTION=0b00000011;          /* assign prescaler to T0 */

/* initialize INTCON register (keep GIE inactive!) */
    INTCON=0b00000000;          /* disable all interrupts */

/* initialize PIE1 and PIE2 registers (periphreal interrupts) */
    PIE1=0b00000000;            /* disable all peripheral interrupts */
    PIE2=0b00000000;

/* initialize T1CON and T2CON registers */
    T1CON=0b00000000;           /* T1 not used  */
    T2CON=0b00000000;           /* T2 not used  */

/* initialize CCPxCON registers */
    CCP1CON=0b00001100;         /* set CCP1CON for PWM mode */
    CCP2CON=0b00000000;         /* CCP2CON=0 (PWM 2 not used) */

/* initialize SSPCON register */
    SSPCON=0b00000000;          /* serial port - not used */

/* initialize ADCONx registers */
    ADCON0=0b00000000;          /* a-d converter */
    ADCON1=0b00000010;

/* initialize TRISx register (port pins as inputs or outputs) */
    TRISA=0b00001111;
    TRISB=0b00000000;
    TRISC=0b11111011;
    TRISD=0b11111111;
    TRISE=0b11111111;

/* clear watchdog timer (not used) */
    CLRWDT();

/* initialize program bit variables */
   FLAGS=0b00000000;

/* intialize output port pins */
   PORTB=0;

/* enable interrupts...  */

    INTCON.T0IE=1;              /* peripheral interrupt enable */
    INTCON.GIE=1;               /* global interrupt enable */

    init_PWM(0x40);             /* init PWM port */
    init_filter();              /* init filter */
    while(1){
      while(!sample_flag){}     /* wait for sample clock flag to be set */
        sample_flag=0;          /* clear sample clock flag */
        x_n=get_sample(1);      /* read ADC channel 1 into x(n) */
        if(FILTER==1){          /* if filter enabled */
          filter();             /* call filter routine */
        }
        else{                   /* or else write x(n) to y(n) (talkthru) */
          y_n=x_n;
        }
        write_PWM((char)y_n,0); /* write y_n to PWM port 1 */
    }
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX F: 8-BIT MULTIPLY AND SCALING ROUTINES

```
/*************************************************************************
* 8 bit Multiply and Scaling Routines
*
* Written for "Digital Signal Processing with the PIC16C74" Application Note.
*
*
* This module provides a 8 bit signed multiply and scaling routine for the
* PICTONE.C tone generation program. The routines are adapted from "Math
* Routines for the 16C5x" in  Microchip's Embedded Controller Handbook.
*
* All numbers are assumed to be signed 2's compliment format.
*
* D. Mostowfi 11/94
*************************************************************************/
char multcnd;                   /* 8 bit multiplicand */
char multplr;                   /* 8 bit multiplier */
char result_h;                  /* result - high byte */
char result_l;                  /* result - low byte */
char sign;                      /* result sign */

#asm

;
; 8x8 signed multiply routine
; called from PICTONE.C module (assembly language routine)
;
 .MACRO mult_core bit
        btfss   multplr,bit
        goto    \no_add
        movf    multcnd,W
        addwf   result_h,F

\no_add:
        rrf     result_h
        rrf     result_l
 .ENDM


_8x8smul:
        movf    multcnd,W       ; get multiplicand
        xorwf   multplr,W       ; and xor with multiplier
        movwf   sign            ; and save sign of result
        btfss   multcnd,7       ; check sign bit of multiplicand
        goto    chk_multplr     ; go and check multipier if positive
        comf    multcnd         ; negate if negative
        incf    multcnd         ;

chk_multplr:
        btfss   multplr,7       ; check sign bit of multiplier
        goto    multiply        ; go to multiply if positive
        comf    multplr         ; negate if negative
        incf    multplr         ;

multiply:
        movf    multcnd,W       ; set up multiply registers
        bcf     STATUS,C        ;
        clrf    result_h        ;
        clrf    result_l        ;
        mult_core 0             ; and do multiply core 8 times
        mult_core 1             ;
```


© 1997 Microchip Technology Inc.

```
        mult_core 2             ;
        mult_core 3             ;
        mult_core 4             ;
        mult_core 5             ;
        mult_core 6             ;
        mult_core 7             ;

set_sign:
        btfss   sign,7          ; test sign to see if result negative
        retlw   0               ; done if not! (clear W)
        comf    result_l        ; negate result if sign set
        incf    result_l        ;
        btfsc   STATUS,Z        ;
        decf    result_h        ;
        comf    result_h        ;

        retlw   0               ; done (clear W)


;
; Scaling Routine (used after a multiply to scale 16 bit result)
; Operates on result_h and result_l - final result is in result_l
; routine divides by 32 to restore Q7 result of 2*b*y(n-1) in tone
; generation algorithm
;
scale_16:
        btfss   sign,7          ; test if negative (sign set from mult)
        goto    start_shift     ; go to start shift if pos.
        comf    result_l        ; negate first if neg.
        incf    result_l        ;
        btfsc   STATUS,Z        ;
        decf    result_h        ;
        comf    result_h        ;

start_shift:
        bcf     STATUS,C        ; clear status
        rrf     result_h        ; and shift result left 5x (/32)
        rrf     result_l        ;
        rrf     result_h        ;
        rrf     result_l        ;
        rrf     result_h        ;
        rrf     result_l        ;
        rrf     result_h        ;
        rrf     result_l        ;
        rrf     result_h        ;
        rrf     result_l        ;

        btfss   sign,7          ; test if result negative
        goto    scale_done      ; done if not negative
        comf    result_l        ; negate result if negative
        incf    result_l        ;
        btfsc   STATUS,Z        ;
        decf    result_h        ;
        comf    result_h        ;

scale_done:                     ;

        retlw   0               ; done (clear W)

#endasm
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX G: DOUBLE PRECISION MATH ROUTINES

```
/*************************************************************************
* Double Precision Math Routines
*
* This module contains assembly language routines from "Math Routines for the
* 16C5x" from Microchip's Embedded Controller Handbook that have been adapted
* for use with the Bytecraft MPC C Compiler.
*
* Routines are used IIR_FILT.C module written for "Digital Signal Processing
* with the PIC16C74" Application Note.
*
* D. Mostowfi 3/95
*************************************************************************/

/*
Start of converted MPASM modules:

;*****************************************************************
;                 Double Precision Addition & Subtraction
;
;*****************************************************************;
;   Addition :  ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
;       (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
;       (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
;       (c) CALL D_add
;       (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
;
;   Performance :
;               Program Memory  :       07
;               Clock Cycles    :       08
;*****************************************************************;
;   Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
;       (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
;       (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
;       (c) CALL D_sub
;       (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
;
;   Performance :
;               Program Memory  :       14
;               Clock Cycles    :       17
;*****************************************************************;
;
*/

char ACCaLO;  //equ     10   changed equ statements to C char variables
char ACCaHI;  //equ     11
char ACCbLO;  //equ     12
char ACCbHI;  //equ     13
;

#asm            /* start of in-line assembly code */

;       include "mpreg.h"     commented out these
;       org     0             two lines (MPASM specific)

;*****************************************************************
;         Double Precision Subtraction ( ACCb - ACCa -> ACCb )
;
D_sub   call    neg_A2          ; At first negate ACCa; Then add
;
;*****************************************************************
```

```
;       Double Precision  Addition ( ACCb + ACCa -> ACCb )
;
D_add   movf    ACCaLO,W
    addwf   ACCbLO              ;add lsb
    btfsc   STATUS,C            ;add in carry
    incf    ACCbHI
    movf    ACCaHI,C
    addwf   ACCbHI              ;add msb
    retlw   0

neg_A2  comf    ACCaLO          ; negate ACCa ( -ACCa -> ACCa )
    incf    ACCaLO
    btfsc   STATUS,Z
    decf    ACCaHI
    comf    ACCaHI
    retlw   0


;*****************************************************************
;                       Double Precision Multiplication
;
;               ( Optimized for Speed : straight Line Code )
;
;*****************************************************************;
;   Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCb,ACCc ( 32 bits )
;       (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
;       (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
;       (c) CALL D_mpy
;       (d) The 32 bit result is in location ( ACCbHI,ACCbLO,ACCcHI,ACCcLO )
;
;   Performance :
;               Program Memory  :       240
;               Clock Cycles    :       233
;
;       Note : The above timing is the worst case timing, when the
;               register ACCb = FFFF. The speed may be improved if
;               the register ACCb contains a number ( out of the two
;               numbers ) with less number of 1s.
;
;               The performance specs are for Unsigned arithmetic ( i.e,
;               with "SIGNED equ  FALSE ").

;*****************************************************************;
;

#endasm
//char ACCaLO;  equ     10   Commented out - already defined in Dbl_add
//char ACCaHI;  equ     11
//char ACCbLO;  equ     12
//char ACCbHI;  equ     13
char ACCcLO;  //equ     14   changed equ statements to C char variables
char ACCcHI;  //equ     15
char ACCdLO;  //equ     16
char ACCdHI;  //equ     17
char temp;    //equ     18
char sign;    //equ     19

#asm
;
;       include "mpreg.h"       commented out these
;       org     0               two lines (MPASM specific)
;*****************************************************************
SIGNED  equ     1               ; Set This To 'TRUE' if the routines
;                               ; for Multiplication & Division needs
;                               ; to be assembled as Signed Integer
;                               ; Routines. If 'FALSE' the above two
```


© 1997 Microchip Technology Inc.

```
;                               ; routines ( D_mpy & D_div ) use
;                               ; unsigned arithmetic.
;*****************************************************************
;       multiplication macro
;
 .MACRO mulMac                  ; changed macro to conform to MPC macro
;        LOCAL   NO_ADD         ; language - declaration is different
;                               ; and macro labels are preceded by "/"

    rrf     ACCdHI              ; rotate d right
    rrf     ACCdLO
    btfss   STATUS,C            ; need to add?
    goto    \NO_ADD             ; no addition necessary
    movf    ACCaLO,W            ; Addition ( ACCb + ACCa -> ACCb )
    addwf   ACCbLO              ; add lsb
    btfsc   STATUS,C            ; add in carry
    incf    ACCbHI
    movf    ACCaHI,W
    addwf   ACCbHI              ;add msb
\NO_ADD rrf     ACCbHI
    rrf     ACCbLO
    rrf     ACCcHI
    rrf     ACCcLO
;
    .ENDM                       ; end of modified macro
;
;*****************************************************************;
;               Double Precision Multiply ( 16x16 -> 32 )
;         ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
;  in ACCb ( ACCbHI,ACCbLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
;
D_mpyF                          ;results in ACCb(16 msb's) and ACCc(16 lsb's)
;
     .IF   SIGNED
     CALL    S_SIGN
     .ENDIF
;
    call    setup
;
; use the mulMac macro 16 times
;
    mulMac
    mulMac
    mulMac
    mulMac
    mulMac
    mulMac
    mulMac
    mulMac
    mulMac
    mulMac
    mulMac
    mulMac
    mulMac
    mulMac
    mulMac
    mulMac
;
    .IF    SIGNED
    btfss   sign,7
    retlw   0
    comf    ACCcLO              ; negate ACCa ( -ACCa -> ACCa )
    incf    ACCcLO
    btfsc   STATUS,Z
    decf    ACCcHI
    comf    ACCcHI
```

```
    btfsc   STATUS,Z
neg_B   comf    ACCbLO          ; negate ACCb
    incf    ACCbLO
    btfsc   STATUS,Z
    decf    ACCbHI
    comf    ACCbHI
    retlw   0
    .ELSE
    retlw   0
    .ENDIF
;
;*****************************************************************
;
setup   movlw   16              ; for 16 shifts
    movwf   temp
    movf    ACCbHI,W            ;move ACCb to ACCd
    movwf   ACCdHI
    movf    ACCbLO,W
    movwf   ACCdLO
    clrf    ACCbHI
    clrf    ACCbLO
    retlw   0
;
;*****************************************************************
;
neg_A   comf    ACCaLO          ; negate ACCa ( -ACCa -> ACCa )
    incf    ACCaLO
    btfsc   STATUS,Z
    decf    ACCaHI
    comf    ACCaHI
    retlw   0
;
;*****************************************************************
;  Assemble this section only if Signed Arithmetic Needed
;
     .IF    SIGNED
;
S_SIGN  movf    ACCaHI,W
    xorwf   ACCbHI,W
    movwf   sign
    btfss   ACCbHI,7            ; if MSB set go & negate ACCb
    goto    chek_A
;
    comf    ACCbLO              ; negate ACCb
    incf    ACCbLO
    btfsc   STATUS,Z
    decf    ACCbHI
    comf    ACCbHI
;
chek_A  btfss   ACCaHI,7        ; if MSB set go & negate ACCa
    retlw   0
    goto    neg_A
;
     .ENDIF

#endasm
```


© 1997 Microchip Technology Inc.

PIC14C000 Calibration Parameters

Author: Richard Fisher
Microchip Technology Inc.

Caution:

When using a WINDOWED DEVICE, the calibration data will be lost if the device is erased. Care should be exercised to ensure that factory calibration data is not lost. Microchip does not recommend using code protection in a windowed device.

Due to the photosensitivity of semiconductors, the analog peripherals may not function properly if the window protection label is not in place during device operation.

INTRODUCTION

This application note discusses the PIC14C000 calibration constants and calibration procedures.

This application note refers to the following documents:

- PIC14C000 Data Sheet (DS40122)
- AN575, "IEEE 754 Compliant Floating Point Routines" (DS00575)
- AN624 "PIC14C000 A/D Theory and Implementation" (DS00624)

The PIC14C000 has several analog peripherals. Like all CMOS circuitry the parametric values vary with process, temperature, voltage, and time.

The PIC14C000 has been designed to minimize the effect of these variations. In addition, each device is calibrated at factory test by measuring several key parameters and storing these values into EPROM at specified locations. The customer's application program may access this data and use it to mathematically compensate for device variations.

Collectively, these data values are referred to as calibration constants. They are listed in Table 4-1 in the PIC14C000 Data Sheet (DS40122).

CALIBRATION DATA

The PIC14C000 calibration constants are listed in Table 1.

TABLE 1: CALIBRATION CONSTANTS

Parameter	Symbol
A/D Slope reference ratio	KREF
Bandgap reference voltage	KBG
Temperature sensor voltage	VTHERM
Temperature sensor voltage slope	KTC
Internal oscillator frequency multiplier	FOSC
Watchdog Timer (WDT) time-out	TWDT

Table 2 show an example of PIC14C000 calibration constants and their locations.

The first four parameters in the table are in 32-bit floating point representation. Each parameter has an exponent byte, and three bytes of mantissa. (For information on floating point algorithms, refer to AN575.) The last two parameters are single-byte numbers.

TABLE 2: TYPICAL CALIBRATION CONSTANT VALUES

Constant	Address	Value	Byte 1	Byte 2	Byte 3	Byte 4
KREF	0FC0:0FC3	0.1259	7C	00	F8	DD
KBG	0FC4:0FC7	1.1842	7F	17	93	02
VTHERM	0FC8:0FCB	1.0766	7F	09	CC	CD
KTC	0FCC:0FCF	0.0037	76	75	B3	3C
FOSC	0FD0	4.65	A5	—	—	—
TWDT	0FD2	16	10	—	—	—

Note: For the 4-byte constants, Byte 1 is the exponent, while Bytes 2, 3, and 4 specify Mantissa High Byte, Middle Byte, and Low Byte, respectively. For the 1-Byte constants, the value occupies only Byte 1.

PRO MATE is a registered trademark of Microchip Technology Inc.

© 1997 Microchip Technology Inc.

USING CALIBRATION DATA

The calibration constants should be used by the application firmware to obtain the best accuracy.

KREF and KBG are used in A/D conversions. For further details, see AN624.

VTHERM and KTC are used for the internal temperature sensor. To use these parameters to calculate temperature, see AN624.

FOSC is a 1-byte value that represents the measured internal oscillator frequency. The internal oscillator frequency can be calculated as follows:

$$f = \left(3 + \frac{FOSC}{100}\right) MHz$$

FOSC can be used to compensate device internal clock variations for time critical applications such as USART baud rate generation.

TWDT is a 1-byte value that is the watchdog time-out in milliseconds without using the postscaler counter. The postscaler counter can extend the WDT time out up to 2.5 seconds. See the PIC14C000 Data Sheet, order number DS40122, for further details.

PARAMETER VARIATION

Table 3 lists the "Maximum Parameter Variation" attainable when the calibration data is not used. This data is based on measurement over the temperature range of -40°C to +85°C and over the operating VDD range of 2.7V to 6.0V, unless otherwise specified. For further details, refer to the PIC14C000 Data Sheet.

TABLE 3: MAXIMUM PARAMETER VARIATION WITHOUT CALIBRATION (PRELIMINARY)

Symbol	Parameter	Maximum Variation
KREF	A/D slope reference ratio	+/- 2.2%
KBG	Bandgap reference voltage	+/- 4.2%
VTHERM	Temperature sensor voltage	+/- 12.4% (Note 1)
KTC	Temperature sensor coefficient	+/- 5%
TEMPINT	Calculated temperature	+/- 35°C
TWDT	WDT Time-out Period	+/- 60% (Note 2)

Note 1: At 25°C, +/- 2°C, over VDD range.
2: 7 msec to 33 msec over temperature, VDD = 5.0V

If these accuracies are adequate for the task at hand, no further calibrations are necessary. If greater accuracy is needed, the calibration constants must be used.

Table 4 lists the "Expected Parameter Variation with Calibration." This data is based on measurements using the calibration data (measured at 25°C) over the temperature range of 0°C to +50°C and at the operating VDD = 5.0V unless otherwise specified. For further details, refer to the PIC14C000 Data Sheet.

TABLE 4: EXPECTED VARIATION OF PARAMETERS WITH CALIBRATION (PRELIMINARY)

Symbol	Parameter	Maximum Variation
KREF	A/D slope reference ratio	+/- 0.13%
KBG	Bandgap reference voltage	+/- 0.058%
VTHERM	Temperature sensor voltage	+/- 0.71% (Note 1)
KTC	Temperature sensor coefficient	(NOTE 2)
TEMPINT	Calculated temperature	+/- 2°C (Note 3)
TWDT	WDT Timeout Period	+/- 13.8%

Note 1: At 25° C, +/- 2°C.
2: Calculated as the slope between 25°C and TMAX. TMAX = 70°C for commercial temperature, 85°C for industrial temperature.
3: Refer to AN624 for description of TEMPINT.

© 1997 Microchip Technology Inc.

PROGRAMMING THE PIC14C000

Non-Windowed Parts

Non-windowed parts are programmed just like any PIC16CXXX processor. The calibration area is write-protected during factory calibration and will not be over-written by PRO MATE.

Windowed Parts

Caution:

Windowed parts must not be write-protected. If the parts are erased by ultraviolet light, the calibration parameters are lost and cannot be reprogrammed once the part has been write-protected.

Calibration data must be read out and saved before erasing a windowed part. There is no way to recreate these values, so if they are lost the part can no longer be calibrated.

The calibration data is read by the PRO MATE just like the rest of the code space. After reading, it must be saved as a Hex file. The entire memory space will be saved from 0 to 0xFFF. There is no way to save a part of the memory space.

To erase a windowed part:

1. Read part into the PRO MATE buffer.
2. If 000h - 0FBFh is not blank, use the Fill Prog Buffer command to set all bits to '1'.
3. Save the buffer content as a Hex file.

   Note:

   The calibration data, and hence this file, will be different for each PIC14C000. Don't forget to label the file and the part!
4. Use ultraviolet light to erase the part.
5. Load the saved Hex file (in step 3) into the PRO MATE buffer.
6. Program the part.

The part is now at the same state as it was when the factory shipped it. Load your code and program in the usual fashion.

If desired, after the calibration data file has been loaded to PRO MATE, the application program may be loaded and the whole part programmed at once.

Checksum

The checksum bytes listed in the data sheet are only for use by the PIC14C000 programmer. There is no way for an application program to utilize this checksum data. Note that there are special bytes added to this space so that the checksum of the entire calibration space totals zero. This means that the calibration factors, which vary from part to part, will not affect the user's code checksum, which may be used for program verification.

PROGRAM EXAMPLES

At the end of this application note are two code samples that may be used to access the calibration data. The first reads the entire table and stores the values in RAM. The second program can be used to restore calibration parameters, if accidentally erased.

SUMMARY

This application note has discussed the calibration constants programmed at Microchip Technology Inc. into the PIC14C000. It has discussed the effect of these calibration constants on system accuracy. It has also covered the erasing and reprogramming procedures.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

PIC14C000 CALIBRATION CONSTANTS

```
    list    p=14000, n=74, st=off
    errorlevel -306;suppress reporting of message "Crossing page boundary -- ensure page bits are
set"
                ;from list file

    include p14000.inc

    __CONFIG _CPC_ON & _CPU_OFF & _CPP_OFF & _PWRTE_ON & _WDT_OFF & _FOSC_RC

    ; __CONFIG is a MPASM directive used to embed the Configuration Word setup into.HEX file
    ; See PIC14C000 Data Book for additional information on the Configuration Word

;*************************************************************************************************
;*  The code listed below can be implemented to read the 64 bytes of data located in the PIC14C000
;*  Calibration Space (0FC0h - 0FFFh). The data bytes read will be saved to 64 consecutive RAM
;*  locations starting at location "CAL_ARRAY".
;*
;*************************************************************************************************


; REGISTER EQUATES

        ORG 0x20
TABLE_OFFSET    RES .1                  ;reserve 1 byte for table offset
KREF_PNTR       RES .4                  ;reserve 4 bytes for cal data
KBG_PNTR        RES .4                  ;reserve 4 bytes for cal data
VTHRM_PNTR      RES .4                  ;reserve 4 bytes for cal data
KTC_PNTR        RES .4                  ;reserve 4 bytes for cal data
KIN_PNTR        RES .40                 ;reserve 40 bytes for cal data
CAL_DATA_CHKSUM_PNTRRES.8 ;reserve 8 bytes for cal data

CAL_ARRAY   SET     KREF_PNTR

;*****  COPIES CALIBRATION DATA TO RAM
        ORG     0x00
        clrf    INTCON                  ;ensure INTCON is cleared
        clrf    PCLATH                  ;ensure PCLATH is cleared
        clrf    STATUS                  ;
        goto    start                   ;

        ORG 0x04
isr     goto    isr                     ;do nothing


start   movlw   high  read_cal_table;move high byte of read_cal-table
        movwf   PCLATH                  ;address into PCLATH
        call    read_cal_table     ;
        clrf    PCLATH                  ;reset PCLATH
stop    goto    stop                    ;wait here


        ORG 0xF00
read_cal_table clrf TABLE_OFFSET        ;initialize table offset register
        movlw   CAL_ARRAY               ;call routine for moving cal space data
                                        ;into RAM
        movwf   FSR                     ;initialize RAM array pointer

next_constant movf TABLE_OFFSET,w       ;move array offset value into W Reg.
        call    cal_table               ;get a byte of CAL data
        movwf   INDF                    ;save constant to RAM
```


© 1997 Microchip Technology Inc.

```
        incf    FSR,f                   ;increment RAM cal-array pointer
        incf    TABLE_OFFSET,f          ;point to next calibration data word
        movlw   0x3B                    ;
        subwf   TABLE_OFFSET,w          ;test if all valid cal data has been read
        btfss   STATUS,Z                ;test if result of subtract (compare) is zero
        goto    next_constant           ;go read another table element
        return


        ORG     0xFBF

cal_table   addwf   PCL,f               ;Add table offset to PC

        END
```


© 1997 Microchip Technology Inc.

CALIBRATION DATA TABLE

```
    list    p=14000,n=74,st=off
    errorlevel -306;suppress reporting of message "Crossing page boundary -- ensure page bits are
set"
                ;from list file

    include p14000.inc

    __CONFIG _CPC_ON & _CPU_OFF & _CPP_OFF & _PWRTE_ON & _WDT_OFF & _FOSC_RC

    ; __CONFIG is a MPASM directive used to embedd the Configuration Word setup into.HEX file
    ; See PIC14C000 Data Book for additional information on the Configuration Word

;*******************************************************************************************
;* TITLE:   Calibration data table
;* PURPOSE: Create a hex file of the calibration data to program into the PIC14C000
;* DESCRIPTION:
;*  This table is only needed if you erase, accidentally or otherwise, the existing
;*  calibration data that was preprogrammed in the PIC14C000.
;*    In order to get the calibration data, you have to read the PIC14C000 (before
;* you erase it) and save the data in a hex or text file.  So, it is a good idea
;* to read the cal data and save it before you start your development work.
;*
;*******************************************************************************************


        ORG 0xFC0                   ;Start of calibration data

;           ----------------------------------------Microchip modified IEEE754
;           |     | mantissa | mantissa | mantissa |32-bit floating point format
;           |exp  | hi byte  | mid byte | low byte |(see appnote AN575)
;           ----------------------------------------
Kref    DT   0x7B,    0x64,       0xC3,       0xEE   ;A/D Slope Ref.
Kbg     DT   0x7F,    0x13,       0x35,       0x40   ;Bandgap Ref. Voltage
Vthrm   DT   0x7F,    0x07,       0xC8,       0xB5   ;Temperature Sensor Voltage
Ktc     DT   0x76,    0x75,       0xB3,       0x3C   ;Volts / (degree Celsius)
Kin     DT   0x69                                    ;Fosc (unsigned byte)

fill_startFILL(retlw 0xFF), (CAL_DATA_CHKSUM - fill_start);FD1 - FF8 = retlw 0xFF


        org     0xFF9               ;cal data checksum address

CAL_DATA_CHKSUM                     ;calibration data checksum FC0H-FF8H
        DT      0xC3,0xB5
        FILL    0x3FFF,3            ;next three words are not unused
        org     0xFFE

        DW      0x12D6              ;special checksum data
                                    ;makes calibration space (FC0H-FCDH)
                                    ;checksum equal 0000H.  Hence,
                                    ;the cal data has no effect on the
                                    ;device checksum calculation

        DW      0x0000              ;Checksum of cal space = 0000H
        end
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN624

PIC14C000 A/D Theory and Implementation

Author: Brian Dellacroce
Design Consultant

INTRODUCTION

This application note presents an implementation of a data acquisition system based on a PIC14C000 microcontroller. The PIC14C000's analog front end will perform signal conditioning and analog to digital conversion of signals used in a battery monitoring application. Battery voltage, current, and temperature data will be acquired. This data will then be transmitted via RS-232 to a remote PC host.

SLOPE A/D CONVERSION

With the addition of an external capacitor, the analog front end of the PIC14C000 can be used as a slope A/D converter. The slope A/D is formed when a constant current source (CDAC pin) drives a capacitor. From basic capacitor theory, the voltage across this capacitor is described by the equation:

EQUATION 1: CAPACITOR VOLTAGE

$$V(t) = 1/C \times \int_{t0}^{t1} i(t)dt$$

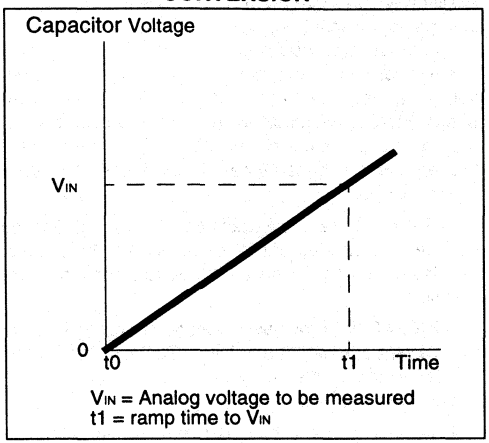
Since the current $i(\tau)$ is constant, this reduces to:

EQUATION 2: VOLTAGE RAMP

$$V1 - V0 = 1/C \times I(t1 - t0)$$

This relationship describes a basic linear voltage ramp. The conversion from analog to digital occurs by measuring the time from the start of the linear ramp (*t0*) to the time (*t1*) that the voltage across the capacitor equals the analog input voltage to be converted. This time is measured by a digital counter circuit. Thus, the delta time (*t1 - t0*) value represents the magnitude of the analog voltage being converted. In this application note, the measured delta time value will simply be referred to as the "count" for the particular analog voltage being converted.

FIGURE 1: LINEAR VOLTAGE RAMP A/D CONVERSION



The magnitude of the measured analog voltage can be determined if the capacitance and the charging current are known. However, in practice, these values cannot be exactly determined. Therefore, a known reference voltage is used instead. This reference voltage is supplied by a very stable circuit known as a bandgap reference. The analog input voltage is determined ratiometrically by performing two successive A/D conversions, the first on the analog input voltage, and the second on the reference voltage. The analog voltage being measured can be calculated using the equation below.

EQUATION 3: VOLTAGE MEASURE

$$V_{IN} = \frac{N_{IN}}{N_{BG}} \times K_{BG}$$

N_{IN} = A/D count value for selected input.

N_{BG} = A/D count value for bandgap reference.

K_{BG} = Absolute voltage value of the bandgap reference voltage, stored in calibration space EPROM.

Equation 3 assumes that the starting voltage of the linear ramp (*V0*) in both the analog input and reference voltage conversions is zero. In practice, however, the capacitor cannot be discharged completely from the prior conversion and a few millivolts worth of charge,

© 1997 Microchip Technology Inc.

referred to as offset voltage, may remain on the capacitor. This effect is called capacitor dielectric absorption, and varies depending on the capacitor's dielectric material, the voltage to which it was charged during the last charge cycle, and the amount of time the capacitor has had to discharge. While teflon capacitors exhibit the lowest dielectric absorption, polystyrene and polyethylene are also very good. Ceramic, glass and mica are fair, and tantalum and electrolytic types are poor choices for A/D applications.

Additionally, the comparator (that compares the ramp voltage to the input voltage and stops the timer when the voltages are equal) usually has a few millivolts of offset error in its comparison. This comparator offset voltage adds (or subtracts) to the dielectric absorption offset. Finally, the counter that times the conversion may have a very small constant turn-on or turn-off delay that affects the measurement in the same manner as offset voltage.

In order to calculate an exact value for the analog input voltage, the various offset voltages must be factored into the equation. To do this, they are combined and measured.

The PIC14C000 provides two slope reference voltages for this purpose: upper and lower. The ratio of the magnitudes of these references is known and can be used to calculate the combined offset voltage.

This is done by performing conversions on both slope references, and using the supplied calibration constant for the ratio of their magnitudes, and calculating the offset voltage as an offset count. This offset count is then used in the other equations.

The equation used is that of calculating the x-intercept of a straight line when the slope of the line and two x coordinates are known.

EQUATION 4: OFFSET COUNT

$$Noffset = Nreflo - Kref(Nrefhi - Nreflo)$$

Where:

N_{REFLO}	=	A/D count value low reference point.
N_{REFHI}	=	A/D count value high reference point
K_{REF}	=	S_{REFLO} / (S_{REFHI} - S_{REFLO}), stored in calibration space in EPROM.
S_{REFHI}	=	See PIC14C000 Data Sheet, section 8.
S_{REFLO}	=	See PIC14C000 Data Sheet, section 8.

This offset term and its measurement are illustrated graphically in Figure 2. Applying this offset term to the calculation of V_{IN} compensates the count obtained for both the input channel and the reference channel, to yield the counts as if each conversion did start from zero, instead of some offset voltage. Thus, the following equation may be used to determine the voltage at any input:

EQUATION 5: VOLTAGE AT ANY INPUT

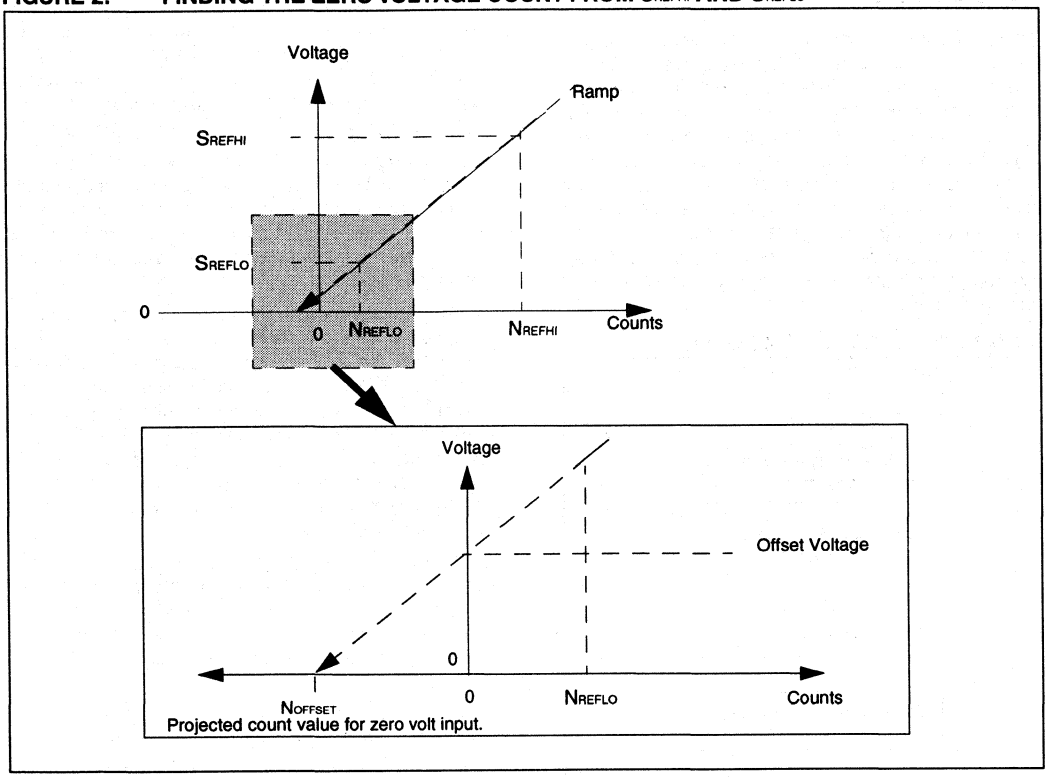
$$V_{IN} = \frac{N_{IN} - N_{OFFSET}}{N_{BG} - N_{OFFSET}} \times K_{BG}$$

RA1/AN1 and RD5/AN5 Analog Input Pins

The AN1 and AN5 inputs pins have a special input offset feature that can allow these inputs to measure voltages that are either above or below device ground potential. This is accomplished with a summing junction that adds 0.5V to the signal at the pin before it reaches the A/D comparator. In order to determine the absolute voltage at the pin if the offset feature is turned on, two A/D conversions must be made. First, the input combined with the summing signal (0.5V) is converted. Then, a second conversion is performed with the AN1/AN5 side of the summing junction disconnected, resulting in just the 0.5V summing signal being converted. To compute the value of the voltage at the AN1/AN5 pin, the result from the second conversion is subtracted from the result of the first conversion.

© 1997 Microchip Technology Inc.

FIGURE 2: FINDING THE ZERO VOLTAGE COUNT FROM S_{REFHI} AND S_{REFLO}

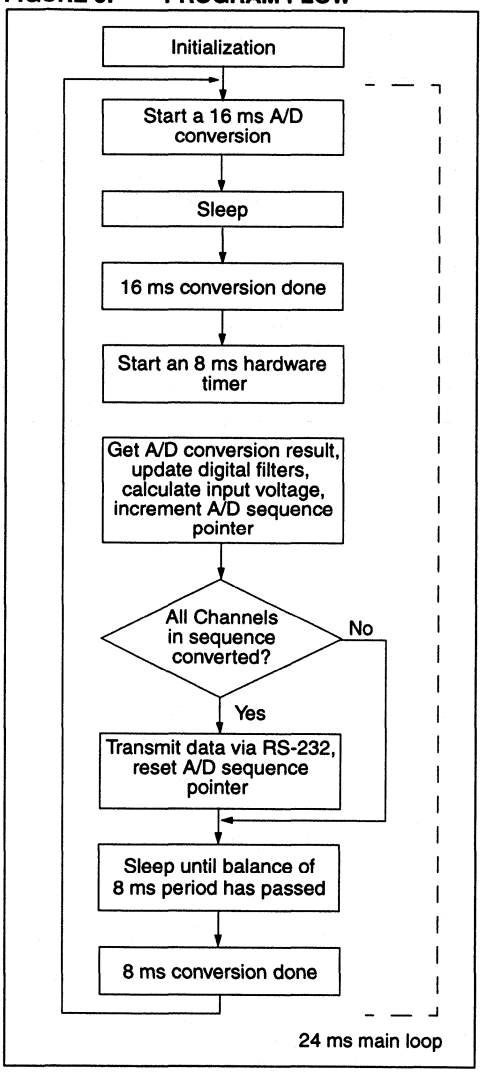


© 1997 Microchip Technology Inc.

IMPLEMENTATION

After initilization, the program continuously executes a 24 ms main loop. 16 ms is allocated for each A/D conversion, and 8 ms is allotted for calculations and other tasks. In order to conserve power and reduce noise in the A/D conversions, the CPU is put to sleep during each A/D conversion and during any unused portion of the 8 ms calculation time (Figure 3). This timing scheme allows both the charging time and the discharging time of the voltage ramp capacitor to be constant. This aids in achieving accurate A/D conversions by keeping the offset voltage due to capacitor dielectric absorption constant.

FIGURE 3: PROGRAM FLOW



Hardware Allocation

For this application note, three external analog input channels are used. RA1/AN1, RA2/AN2, and RA3/AN3 are assigned to battery current measurement, battery voltage measurement, and external voltage measurement (in this case a potentiometer), respectively.

Since the A/D converter measures voltage, and not current, the battery current measurement is obtained by measuring the voltage drop across a low value resistor (in this case 0.1 ohm). Actual battery current is then calculated by the firmware using the value of the resistor and Ohms law.

Battery voltage measurement requires that an external resistive voltage divider be used to lower the voltage input into the A/D converter's range. A divider that results in an A/D input voltage that is as near as possible to the bandgap reference voltage is normally selected, since it reduces ratiometric errors during the calculation of the voltage. The firmware calculates the actual battery voltage from the voltage measured at the AN3 pin by multiplying the measured value by the external voltage divider ratio.

The external potentiometer voltage measurement is an auxiliary measurement, and could be used for measuring a thermistor, individual cell voltages, or other parameters. No scaling of this voltage is performed. The program simply reports the voltage sensed at the AN2 pin.

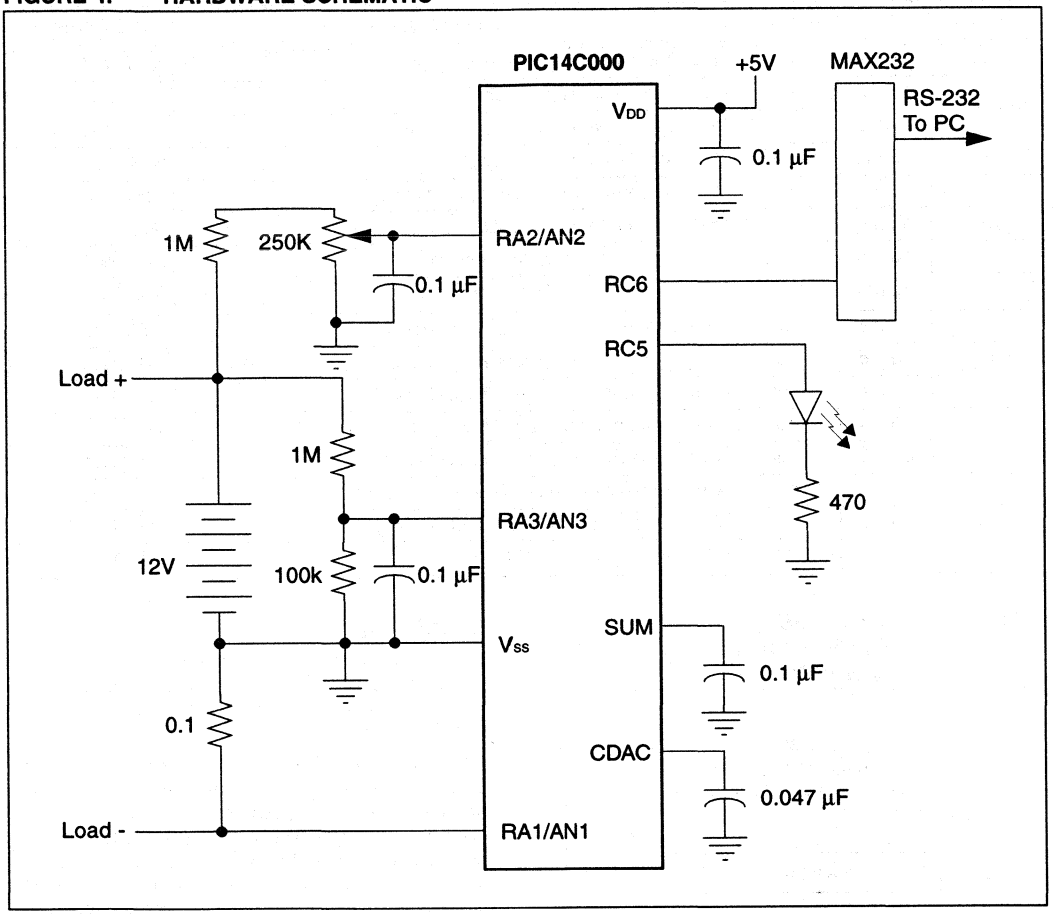
This program assumes that a 0.047 µF capacitor is connected between the CDAC pin of the device and ground. It is used for creating the ramp voltage for the A/D conversion process. Optionally, a 0.1 µF capacitor can be connected between the SUM pin and ground to allow for increased filtering of the battery current (AN1) input.

Port RC6 is used to transmit four parameters: battery current, battery voltage, internal temperature, and AN3 voltage. The format is 9600 baud (8,N,1) RS-232. Each parameter consists of four bytes and is in PICmicro™ 32-bit floating point format. Refer to AN575 for more information on floating point format.

For monitoring purposes, the program will toggle the RC5 output every ~384 ms. It will drive RC5 high just prior to starting the RS-232 data transmission, and drive it low after completing the transmission. If an LED and current limiting resistor are connected to RC5, a visible "heartbeat" indication will appear when the program is running in a normal manner.

© 1997 Microchip Technology Inc.

FIGURE 4: HARDWARE SCHEMATIC



A/D Conversion

The analog to digital conversion firmware modules control the operation of the analog to digital converter hardware. They convert the raw data received from the hardware into physical voltage values. This is accomplished by obtaining readings from the external inputs, the on-chip bandgap reference, and the on-chip A/D reference circuits. The physical voltages at the external inputs are computed based on these readings.

A/D Input Interleaving Sequence

Interleaving the selection of analog inputs scheduled for conversion maximizes the sampling rate for high priority signals such as the battery current, and reduces the sampling rate for low priority signals (that change at relatively slow rates), such as temperature inputs. The following interleaving scheme places the inputs in priority order from highest to lowest. Battery current is the highest priority with eight samples in 16 A/D cycles (~384 ms). Battery voltage is sampled two times in 16 A/D cycles. Battery temperature, via the external thermistor input and internal temperature, is sampled once each during 16 A/D cycles. Current network zero voltage, bandgap voltage, the A/D lower reference voltage, and the A/D upper reference voltage, are each sampled once for every 16 conversion cycles.

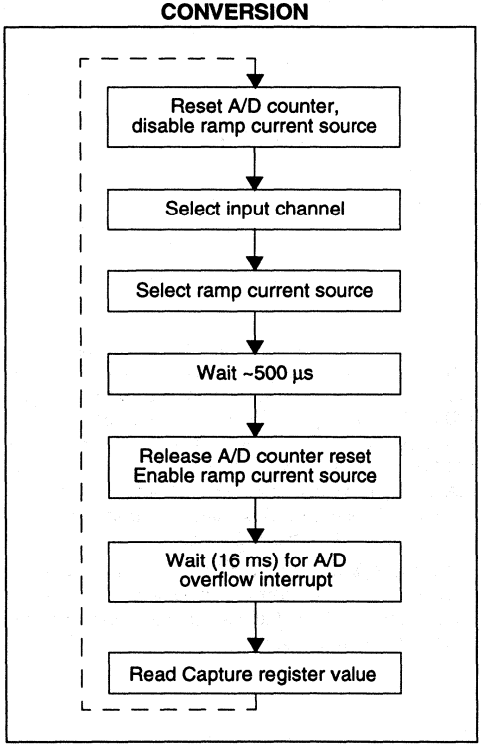
Conversions are scheduled at fixed intervals based upon the maximum conversion time to overflow (16 ms). This fixed scheduling method maintains determinism in the overall system.

© 1997 Microchip Technology Inc.

TABLE 1: CONVERSION OPERATION TABLE

Conv #	T_{REL}	T_{ABS}	Interleave Channel	A/D Channel
0	16 ms	ms	Run sequence on current input channel	AN1: Ibat
1	16 ms	ms	Run sequence on V_{REFHI} channel	S_{REFHI}
2	16 ms	ms	Run sequence on current input channel	AN1: Ibat
3	16 ms	ms	Run sequence on V_{REFLO} channel	S_{REFLO}
4	16 ms	ms	Run sequence on current input channel	AN1: Ibat
5	16 ms	ms	Run sequence on Bandgap channel	V_{BG}
6	16 ms	ms	Run sequence on current input channel	AN1: Ibat
7	16 ms	ms	Run sequence on V_{IN} channel	AN3: Vbat
8	16 ms	ms	Run sequence on current input channel	AN1: Ibat
9	16 ms	ms	Run sequence on current zero channel	AN1: I_{ZERO}
10	16 ms	ms	Run sequence on current input channel	AN1: Ibat
11	16 ms	ms	Run sequence on internal thermistor channel	T_{INT}
12	16 ms	ms	Run sequence on current input channel	AN1: Ibat
13	16 ms	ms	Run sequence on external voltage channel	AN2: V_{EXT}
14	16 ms	ms	Run sequence on current input channel	AN1: Ibat
15	16 ms	ms	Run sequence on V_{IN} channel	AN3: Vbat

FIGURE 5: SEQUENCE FOR PERFORMING A SINGLE A/D CONVERSION



A/D Filtering

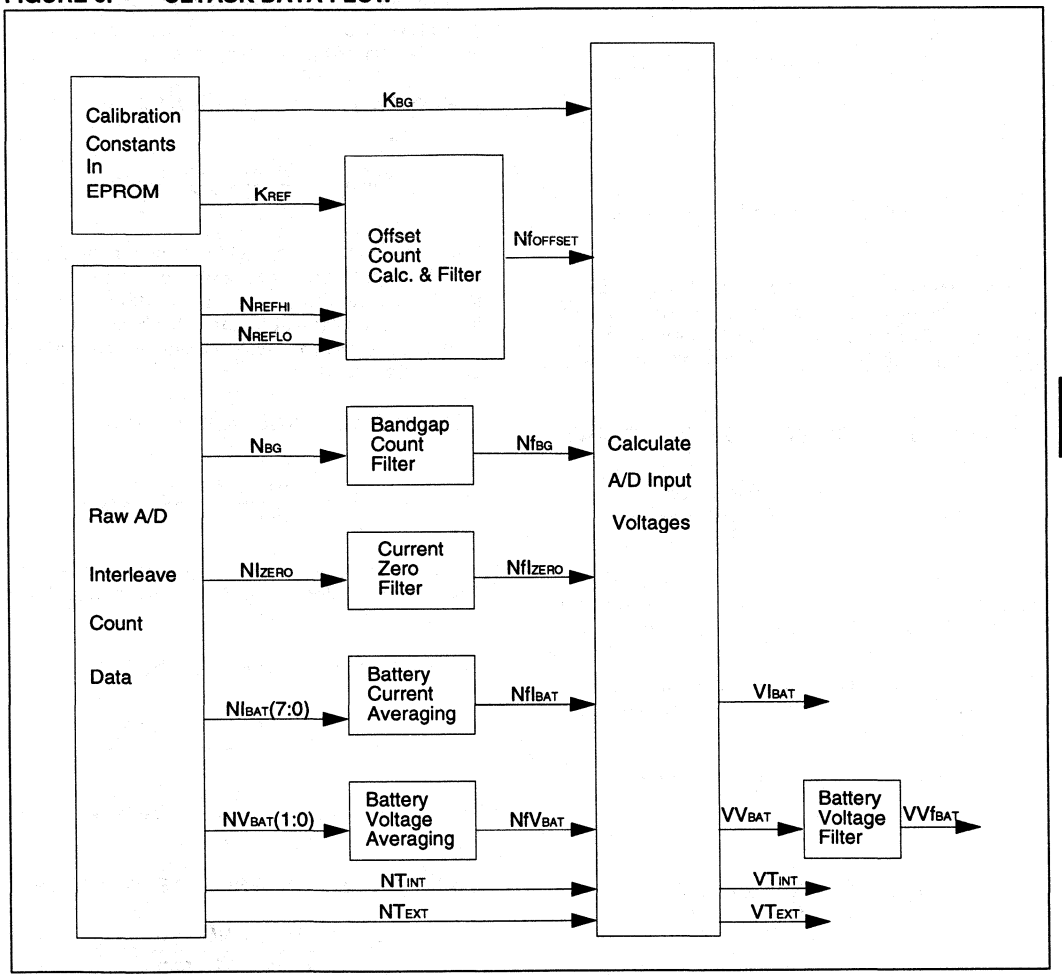
Due to the high resolution of the PIC14C000 A/D converter, digital filtering of the A/D count data is performed. This will reduce noise in the A/D readings, and enhance performance.

A/D count data is converted to PICmicro floating point format prior to being used in any calculation. All results are retained in floating point format as well. DC inputs such as bandgap reference voltage, slope reference high, slope reference low, and current zero offset voltage are filtered with long time-constant filters. Dynamic signals, such as battery voltage and current, are filtered less. Internal temperature and AN3 external voltage are not filtered at all. All calculated outputs are in voltage units. Figure 6 diagrams the data flow through the filtering and calculation task.

The formulas on the following page are used to filter the raw count data obtained from the A/D converter prior to calculating the actual voltage values for the A/D inputs. This filtering stabilizes the reference values to enhance A/D accuracy. The bandgap and offset reference are DC signals, so no information is lost by the filtering process.

© 1997 Microchip Technology Inc.

FIGURE 6: CLTASK DATA FLOW



The count value for the bandgap voltage reference is filtered by calculating the rolling average of the last 16 count values obtained. This filtered value of bandgap count is then used to calculate the input voltages. The subscript i denotes the interleave sequence number, not the conversion number within the interleave sequence.

The count offset value is filtered by calculating the rolling average of the last 16 count values obtained. The filtered value of the offset count is then used in Equation 6 to calculate the input voltages.

EQUATION 6: FILTERED BANDGAP COUNT

$$Nfbg = \frac{NBG_i + 15 \times Nbg_{i-1}}{16}$$

EQUATION 7: FILTERED OFFSET COUNT

$$Nfoffset = \frac{Noffset_i + 15 \times Noffset_{i-1}}{16}$$

where:

$$NOFFSET_i = NREFLO_i - KREF(NREFHI_i - NREFLO_i)$$

The current-input zero offset value is filtered by calculating the rolling average of the last 16 count values obtained. This filtered value of the current-input zero offset is used in the equation to calculate the input voltages.

EQUATION 8: FILTERED ZERO-CURRENT

$$NfIZERO = \frac{NIZERO_i + 15 \times NIZERO_{i-1}}{16}$$

The following formulas are used to filter the raw count data obtained from the battery voltage and current input channels. These filtered values are then used for communication and control functions.

The battery current value is filtered by taking the average of the eight samples of the input channel from the interleave sequence.

EQUATION 9: FILTERED CURRENT COUNT

$$NfIBAT = \left(\frac{1}{8}\right) \sum_{s=0}^{7} (NIBAT_S)$$

The count value for the battery voltage input is filtered by calculating the average of two samples of the input channel from the interleave sequence.

EQUATION 10: FILTERED VOLTAGE COUNT

$$NfVBAT = \left(\frac{1}{2}\right) \sum_{s=0}^{1} (NVBAT_S)$$

Calculation of Input Voltage Values

The following formulas are used to calculate the digital value for each of the input voltages.

EQUATION 11: VOLTAGE AT AN1

$$VIBAT = \frac{NIBAT - NfIZERO}{Nfbg - Nfoffset} \times KBG$$

VIBAT refers to the voltage at the battery current input, AN1. Note that there is no NfOFFSET term in the numerator. The NfOFFSET components of the NfIBAT and NfIZERO terms are canceled out by the subtraction. The equations for the remaining inputs all follow the general form given in Equation 5.

EQUATION 12: VOLTAGE AT AN0

$$VVBAT = \frac{NfVBAT - NfOFFSET}{NfbG - NfOFFSET} \times KBG$$

VVBAT refers to the voltage at the battery voltage input, AN0. Additional filtering is added to the computed VVBAT signal using the equation:

EQUATION 13: FILTERED VOLTAGE

$$VVfBAT = \frac{VVBAT_i + 7 \times VVbat_{i-1}}{8}$$

Temperature terms are calculated directly without any added filtering.

EQUATION 14: INTERNAL TEMPERATURE

$$VTINT = \frac{NTINT - NfOFFSET}{NfbG - NfOFFSET} \times KBG$$

VTINT refers to the voltage at the internal temperature sensor.

EQUATION 15: EXTERNAL TEMPERATURE

$$VTEXT = \frac{NText - NfOFFSET}{NfbG - NfOFFSET} \times KBG$$

VTEXT refers to the voltage at the external temperature sensor input.

© 1997 Microchip Technology Inc.

Conversion to Physical Units

The physical unit values (i.e., temperature, battery voltage, battery current) for the input voltages are computed prior to being transmitted via RS-232. To reduce data storage requirements, these values are not maintained in memory, but are destroyed after they are transmitted. Scaling constants read from EPROM are used to convert the voltage values measured at the pin into the appropriate physical value. The voltage at the AN2 pin is not scaled prior to transmission; the actual voltage at the pin is transmitted.

Current

$$I_{BAT} = \frac{V_{IBAT}}{R_{ISENSE}}$$

I_{BAT} = Battery current.

V_{IBAT} = measured voltage at RA1/AN1 pin.

R_{ISENSE} = External current sense resistor value (stored in user EPROM).

Voltage

$$V_{BAT} = VVf_{BAT}(V_{BAT}R_{DIV})$$

V_{BAT} = Battery voltage.

VVf_{BAT} = Filtered measured voltage at RA3/AN3 pin.

R_{DIV} = External resistor divider ratio for RA2/AN2 pin (stored in user EPROM).

Temperature

$$T_{EMPINT} = \frac{VT_{INT} - Vtherm}{Ktc} + 298.15$$

T_{EMPINT} = Temperature in degrees Kelvin.

VT_{INT} = measured temperature sensor voltage value.

V_{THERM} = Factory calibrated voltage value of the temperature sensor at 298 Kelvin (stored in calibration place in EPROM)

K_{TC} = Factory calibrated temperature constant of temperature sensor in volts per degrees Kelvin (stored in calibration space in EPROM).

RS-232 COMMUNICATION

The physical battery current, voltage, temperature, and RA3/AN3 voltage values are transmitted on the RC6 pin in 8,N,1 RS-232 format.

The transmit routine is designed to make use of the PIC14C000's calibrated internal (IN) oscillator. The oscillator frequency, stored in the factory calibration EPROM, is used as an index into a bit-delay lookup table. The bit-delay value controls the number of instructions executed during the software delay routine and is used to generate the time between bits being transmitted. The time resolution of this lookup table method results in a worst- case bit time error of less than two percent.

Several look up tables for baud rate are included as assembly time options. Normally the 9600 baud option is used.

ASSEMBLY FILES

The program code for this application note is available on the Microchip BBS as the AN624.ZIP file. Once the file is unzipped, the user should read the README file for further instructions.

© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN626

Lead-Acid Battery Charger Implementation Using PIC14C000

Author: *Dan Butler*
Microchip Technology Inc.

INTRODUCTION

The PIC14C000 comes with several peripherals specifically aimed at the battery market. The programmable reference and onboard comparators are useful for creating charge control circuits, while the analog-to-digital (A/D) converter can monitor the charge state to prevent overcharge. The control software is written in "C" for maintainability and transportability. Where necessary, in line assembly is used.

This application note is intended to demonstrate the use of the PIC14C000 in an intelligent battery charger. The charger is designed to charge a sealed lead-acid battery (YUASA NP7-12 12V, 7AH); however, the charge parameters are easily modified to work with different lead-acid batteries.

The typical method of charging lead-acid batteries is with a constant voltage, current-limited source. That method allows a high initial charge current that tapers off until the battery reaches full charge.

This design uses a constant current, allowing the voltage to rise until the battery voltage reaches a full charge. The charge current is then turned off to prevent overcharging. This allows a high initial charge to quickly bring the battery to a full charge and a low maintenance charge current as needed to maintain the full charge. The constant current design is also easily adaptable to NiCd batteries.

As voltage rises during the charge cycle of the lead-acid battery, it quickly passes 2.1 V/cell. As charging progresses, oxygen begins to be liberated at the positive plates at 2.2 V/cell. At 2.3 V/cell, hydrogen is liberated at the negative plates. This is considered a full charge, as any further current passed into the cell simply releases gasses rather than charging the battery. Hence, the upper voltage limit is set at 13.8V (2.3 V/cell), and the lower voltage is set at 12.6V (2.1 V/cell). As a practical consideration, the lower voltage limit is set slightly lower (12.5V) to lengthen the charge cycles. The battery voltage takes just minutes to decay from over 13.8V to 12.6V. It then takes several hours to decay from 12.6V to 12.5V.

THEORY OF OPERATION

Charge current is controlled by a comparator and programmable reference onboard the PIC14C000. The other side of the comparator is fed by the voltage across a sense resistor. The output of the comparator controls a FET (Q1), which switches the charge voltage on and off. The charge is interrupted once-per-second to read the battery voltage. When it reaches a maximum voltage, the charge current is shut off to prevent overcharge.

The PIC14C000 continues to monitor battery voltage. Over time, the battery voltage decays. When the voltage drops below the lower threshold, the trickle charge is activated to bring the battery back up to full charge.

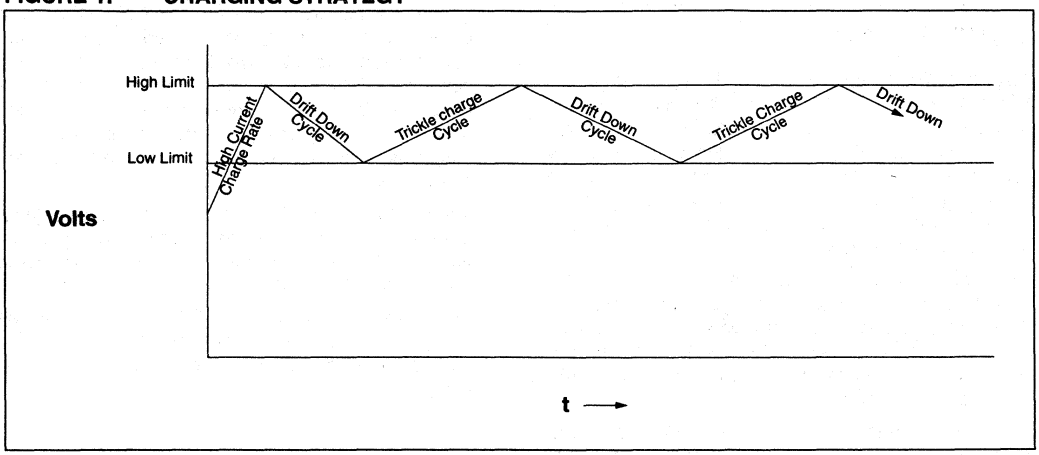
The computing power of the PIC14C000 allows the charger to accurately control the charge cycles to quickly recharge the battery while preventing overcharge.

© 1997 Microchip Technology Inc.

CHARGING STRATEGY

First, charge at a high rate (1A for this battery) until the battery voltage is above the high limit (13.8V). The charge current is then cut off, allowing the battery voltage to decay until it descends past the low limit (12.5V). A low current charge (150 mA) is then applied to again bring the battery voltage up past the high limit. The drift-down/trickle charge cycle repeats (Figure 1).

FIGURE 1: CHARGING STRATEGY



© 1997 Microchip Technology Inc.

ALGORITHM

The controller starts by measuring the voltage on the battery to determine the initial charge rate (high, low or off). Next, it sets up the comparator to control the constant current charge and then goes to sleep.

Note: The comparator continuously controls the current even while the controller sleeps.

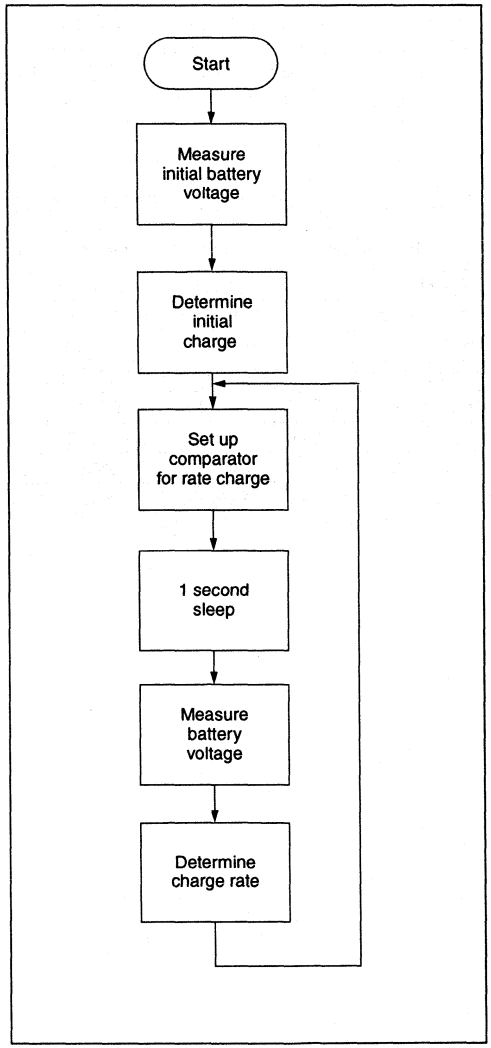
After 1 second, the watchdog timer (WDT) wakes the controller, and the battery voltage is measured again. If any of the trip points are reached, the charge rate is adjusted. After the comparator is reset, the controller goes back to sleep, and the cycle repeats (Figure 2).

Once the measurement/decision cycle is complete, the controller goes to sleep for about 1.15 seconds (subject to the drift of the WDTs internal RC oscillator). Time-out of the WDT wakes up the controller and continues the cycle. The sleep cycle is used to save power and let the hardware do the work of counting the time rather than using timing loops.

Two LEDs are included to provide feedback on what the charger is doing. The red LED signifies a high current charge, and the green LED signifies a low current charge. While the charge is in progress, the active LED blinks at 1 Hz. This is the momentary suspension of charging while the battery voltage is measured.

As the battery ages, it may no longer be able to charge up past the high charge threshold. Time limits have been implemented to account for this. The high charge and low charge cycles have maximum time limits associated with them. The discharge cycle has a minimum time limit.

FIGURE 2: CONTROLLER FLOW CHART



© 1997 Microchip Technology Inc.

Algorithm Parameters:

These parameters are #defines at the top of the code, meaning the code must be recompiled to change the parameters. The hardware could be modified to include dip or rotary switches to change some or all of these parameters.

TABLE 1: CONTROLLER PARAMETERS

Parameter	Units	Range†	Resolution†	Format	Description
V limit low	Volts	0-255.996	.00390625	Fixed point (16:8)	Minimum battery voltage
V limit high	Volts	0-255.996	.00390625	Fixed Point (16:8)	Maximum battery voltage
High current	Amperes	0-255.996	.00390625	Fixed point (16:8)	High charge current
Low current	Amperes	0-255.996	.00390625	Fixed Point (16:8)	Low charge current
High charge time limit	Minutes	0-65536	1	Unsigned long	Maximum time for high charge
Low charge time limit	Minutes	0-65536	1	Unsigned long	Maximum time for low charge
Charge rest time	Minutes	0-65536	1	Unsigned long	Minimum no charge time

† Range and Resolution are the mathematical precision that can be expressed with the parameter, not necessarily what the circuit is capable of.

© 1997 Microchip Technology Inc.

HARDWARE

The PIC14C000 provides two comparators and programmable references (Figure 3). One set is used to maintain the charge current on the battery. Once the comparator is set up, it controls the current without processor intervention. The other comparator is not used in this application; however, it could be used to control current on a second battery to implement parallel charging. Battery voltage is measured using the 16-bit A/D converter.

The board used assumes the existence of an external power supply. This supply needs to provide some headroom above the expected maximum battery voltage and supply enough current for the selected high charge current. In this example, a 16.7V, 2.6A power supply was used.

The charger current to the battery is controlled by the comparator/buck converter. When the comparator senses that the charge current is too high, it pulls the gate of the Q1 low, turning off the current from the power supply and allows current to flow through D2. The buck converter (L1, C2, and D2) takes over and modulates the current to the battery at a controlled rate. When the comparator senses the charge current is too low, it turns on, allowing current from the power supply to flow through Q1. The buck converter now increases current at a controlled rate.

The component values for L1 and C2 are chosen based on the operating parameters of the system. For this system, the buck converter frequency is 15 KHz. The inductor (L1) is calculated from the equation:

$$L = ((V_I - V_{SAT} - V_O)/I_{PK}) \bullet T_{ON}$$

where:

V_I	=	Input Voltage.
V_O	=	Output Voltage.
V_{SAT}	=	Saturation voltage of transistor.
I_{PK}	=	2 Io maximum.
$I_{O\ MAX}$	=	Maximum output current.
T_{ON}	=	"On Time" of duty cycle (output of comparator).

For this design, V_I = 16.7 V, V_{SAT} = 0.25V, V_O = 6.0V (minimum to support 6V battery) I_{PK} = 2A, T_{ON} = 54 µs (80% duty cycle for high current charge):

$$L = ((16.7 - 0.25 - 6.0)/2) \bullet 5\mu s = 282\mu H$$

The output capacitance is chosen such that:

$$C_O \geq I_{PK}T/(8V_{RIPPLE})$$

where:

I_{PK}	=	2 Io maximum
$I_{O\ MAX}$	=	Maximum output current
T	=	Total comparator cycle time
V_{RIPPLE}	=	Output voltage ripple

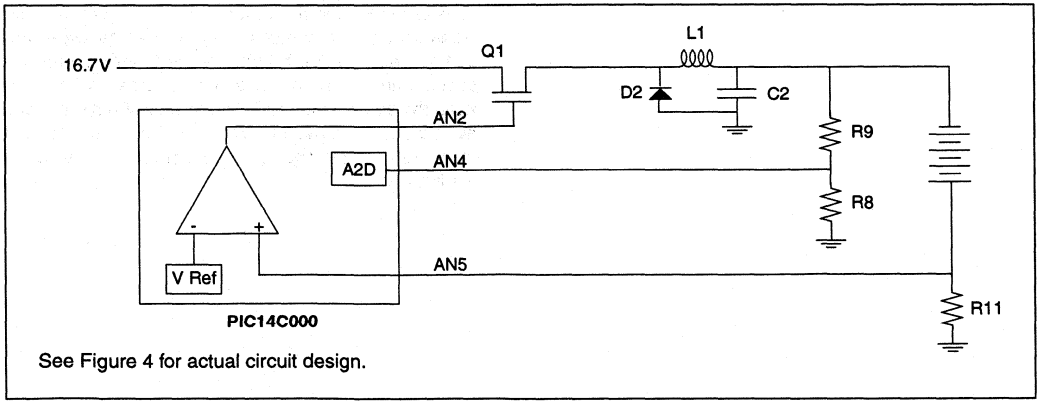
For this design:

$$I_{PK} = 2A, T = 66\mu s, V_{RIPPLE} = 400mV$$

$$C_O \geq (2 \bullet 66)/(8 \bullet 400mV) = 41\mu f$$

The diode (D2) needs to be sized large enough to handle I_{PK}.

FIGURE 3: SIMPLIFIED SCHEMATIC



See Figure 4 for actual circuit design.

DETAILS OF THE SOFTWARE IMPLEMENTATION

Constant Current Control

The comparator is used in conjunction with a programmable voltage reference to control the current into the battery. The voltage reference feeds one side of the comparator, while a sense resistor feeds the other. The output switches a FET to control the current. The voltage reference (PIC14C000 Data Sheet, Section 9 DS40122) is programmed as follows:

$$V = Current \bullet SenseResistor + LevelShift$$

where:

Current	=	The value we want the comparator to control to.
Sense Resistor	=	The value of the current resistor (0.2 Ohms).
Level Shift	=	The 0.5V shift performed on the voltage at the sense resistor (PIC14C000 Data Sheet, Section 9.2 – DS40122B).

This voltage is used in a lookup table which returns the coarse bits (PREFx<7:3>) for the programmable reference. The fine bits (PREFx<2:0>) are calculated as the difference between the voltage and coarse range, divided by the resolution of the table.

Analog-to-Digital Conversion

The battery voltage is measured via the A/D converter. The main program turns off the charger, then runs a conversion on the battery channel. Function AD_Counts performs 16 conversions on the channel, subtracts off the comparator/capacitor offset, and returns the average. The averaging is necessary to remove the noise from the system. The same A/D conversion is also performed on the internal bandgap reference. These values are then used in the following equation from AN624 to obtain the A/D converter voltage:

$$V_{IN} = ((N_{IN} - N_{OFFSET})/(N_{BG} - N_{OFFSET})) \bullet K_{BG}$$

The A/D converter operates most accurately with voltages near the bandgap reference. Therefore, the hardware runs the battery voltage through a voltage divider (R8 and R9) to drop the battery voltage (approximately 13V) down to 1.2V. The battery measurement calculation then multiplies the result from the A/D converter by the resistor ratio to get the original battery voltage:

$$BatteryVoltage = V_{IN} \bullet (R8 + R9)/(R8)$$

Recalibrating the A/D converter (Measuring NOFFSET and NBG) to compensate for component drift is done every cycle. Since the components do not drift very quickly, it's not necessary to recalibrate this frequently, however, it is more accurate and takes advantage of otherwise idle processor time. If processor time is a concern, recalibrating once per minute is sufficient.

Time Keeping

The program counts the seconds to limit the charge cycles on the battery. The WDT times out about every 1.15 seconds, and the rest of the measurement cycle takes about 0.1 seconds, giving have a total cycle time of 1.25 seconds. Each bump of the timer counts for 1 second, and every fourth second another is added to keep the count accurate. This method of timing is only accurate to a few percent. While not good enough for a clock, it's accurate enough to limit the charge cycles on the battery.

© 1997 Microchip Technology Inc.

Math

Fixed point math is used where resolutions of less than one are needed. This code can be updated to use floating point or 32-bit integers, which will allow a cleaner implementation of the calculations required. For this example, limited versions of basic Add/Subtract/Multiply/Divide functions operating on positive 32-bit integers are used. File "MATH32.C" implements 32-bit add, subtract, multiply, divide, and shift. The add, subtract, divide, and shift functions start with 32-bit values and give 32-bit results. The multiply starts with 32-bit multiplicands and gives a 64-bit result.

Charging Circuit Bench Results

When the programmable voltage reference was set to supply the fast charge current of 1A, the actual charge current was measured within 50 mA. However, when the programmable voltage reference was set to supply 150 mA trickle current, the actual output was measured to be as high as 275 mA. This higher trickle current is acceptable for this application since its purpose is to keep the battery topped off at full charge.

This delta is due to design limitations encountered when integrating analog components onto a digital substrate.

Converting the Charger to Work With Other Lead-Acid Batteries

This application note was specifically written to charge a YUASA 12V, 7AH battery, however other batteries may be charged with few modifications. Most of the charge algorithm parameters are in software. However, if the voltage is other than 12V, different resistors (R8 and R9) may be needed in the voltage divider leading to the A/D converter. For example, to charge a 6V, 2AH sealed lead-acid battery requires the necessary changes:

1. Swap R8 for a 137Ω, which will keep the voltage at the A/D converter around the 1.2V optimum.
2. Change the multiplier at the bottom of AN624, Equation 5 to:

$$\frac{(R9 + R8)}{R8} = \frac{(1M + 137K)}{137K} = 8.29927$$

3. Lower the power supply voltage (9V, 1A).
4. Change the charging parameters (#defines at the top of the code):
   a) V_LIMIT_HIGH 7.2V (0x0733 = 7.2 • 256)
   b) V_LIMIT_LOW 6.3V (0x064C = 6.3 • 256)
   c) HIGH_CURRENT (No change)
   d) LOW_CURRENT (No change)
   e) HIGH_CHARGE_TIME_LIMIT 120
   f) LOW_CHARGE_TIME_LIMIT 180
   g) CHARGE_REST_TIME 0

Conclusions

The PIC14C000 has several onboard peripherals that are specifically designed to simplify battery management applications. Further enhancements could be made such as:

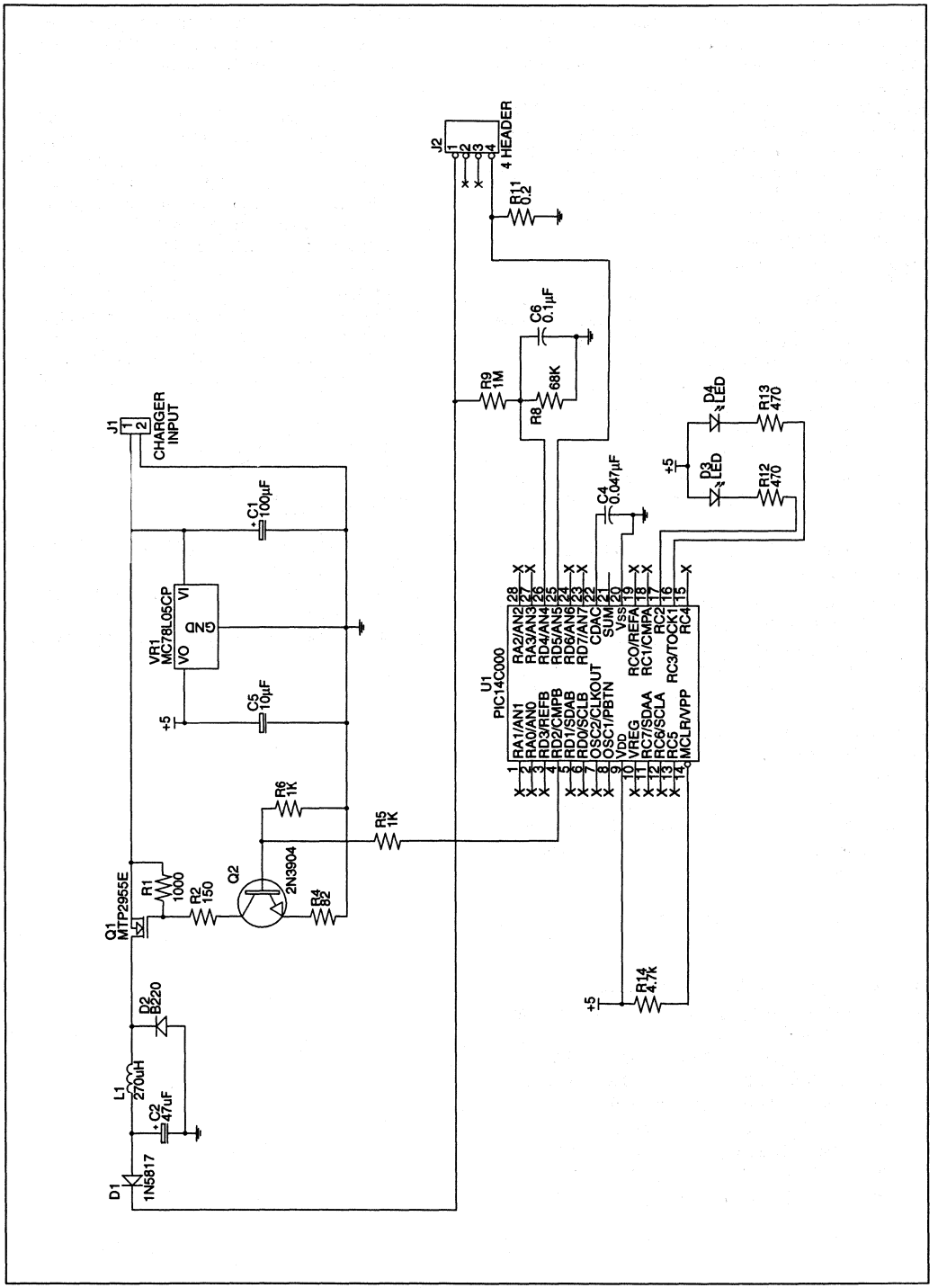
1. Use an onboard temperature sensor to monitor battery temperature, which is another sign of overcharging. Once an over temperature condition is detected, the charge cycle would be terminated regardless off the battery voltage. This, however, would require the PIC14C000 to be physically attached to the battery. Alternatively, a remote temperature sensor could be mounted to the battery and read via the A/D converter.
2. The internal RC oscillator frequency drifts with temperature. This drift rate is known and stored in the calibration data (KTC), which would allow the timer to compensate with more accurate timing.
3. If better charge current accuracy is required, the charging circuitry should be implemented using external components.

REFERENCES

1. PIC14C000 data sheet (DS40122), Microchip Technology, Inc.
2. Microchip AN624 "PIC14C000 A/D Theory and Application," Brian Dellacroce.
3. Battery Reference Book (2nd Ed), T.R. Crompton.

© 1997 Microchip Technology Inc.

FIGURE 4: PIC14C000 LEAD-ACID BATTERY CHARGER



© 1997 Microchip Technology Inc.

PARTS LIST

C1	100 µF	
C2	47 µF	
Q1	MTP2955E	
Q2	2N3904	
C4	0.047 µF	
C5	10 µF	
C6	0.1 µF	
D1	B220	
D2	1N5817	
D3	Green LED	
D4	Red LED	
J1	Connector appropriate to power supply.	
J2	Connector appropriate to battery.	
L1	270 µH	
R1, R5, R6	1K	All Resistors 0.25W unless otherwise specified
R2	150	
R4	82	
R8	68K	
R9	1M	
R11	0.2 Ohm, 1W, wire wound	
R12, R13	470	
R14	4.7K	
U1	PIC14C000	
VR1	7805 Voltage Regulator	

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: LEADACID.C

```
/************************************************************************
*  Filename: LEADACID.C
************************************************************************
*       Author:         Dan Butler
*       Company:        Microchip Technology
*       Revision:       Rev 1.0
*       Date:           29 January 1997
*       Compiler:       MPLAB-C rev 1.10
************************************************************************
*        Include files:
*               14000.h         Version 1.01
*               delay14.h       Version 1.00
*               Math.h          Version 1.00
*               math32.c        32 bit integer math functions
*               timer.c         simple timing functions
*
************************************************************************
*
*  Implements a simple battery charging algorithm.  Uses comparator B to
*  set up a constant current charge.  Takes a reading on the battery once per
*  second to see if the charge is complete.
*
*       Clock Frequency      4 MHz Internal RC
*       Configuration Bit Settings   WDT on, Power up timer off
*       Program and Data Memory Usage
*           Program:  0x4E3
*           Data:     0x7f
*
************************************************************************
*  What's Changed
*
*  Date              Description of Change
*
*
*
************************************************************************/
#include <14000.h>
#include <delay14.h>
#include <math.h>

/* ******************************************************************** */
/* Charging algorithm parameters.   Change these as appropriate for your */
/* application.                                                         */
/* Fixed point values are used below.  To calculate new values, multiply */
/* your floating point value by 2^(number of bits behind the decimal)    */
/* ******************************************************************** */
#define  V_LIMIT_LOW     0x0C80       /* Lower voltage limit 12.50V(2.08V/c)*/
#define  V_LIMIT_HIGH    0x0DCD       /* Upper voltage limit 13.8V(2.3V/c)*/
#define  HIGH_CURRENT    0x0100       /* 0x0100 = 1.000A, Fixed pt (16:8) */
#define  LOW_CURRENT     0x0026       /* 0x0026 = 0.150A, Fixed pt (16:8) */
#define  NO_CURRENT     0             /* off                              */
#define  HIGH_CHARGE_TIME_LIMIT  60   /* Max time for high current charge */
#define  LOW_CHARGE_TIME_LIMIT   120 /* max time for low current charge  */
#define  CHARGE_REST_TIME        0    /* min time for rest between charge */
#define  SENSE_RESISTOR 0x3333        /* .2 ohms, Fixed point 16:16       */
#define  LEVEL_SHIFT     0x8000       /* .5 volt level shift, fp 16:16    */
/* End Algorithm Parameters ************************************************ */

#define  BATTERY_CHANNEL  0xA0        /* battery voltage measured on RD4  */
#define  BAND_GAP_CHANNEL 0x40      /* Band Gap Reference */
```


© 1997 Microchip Technology Inc.

```
#define  TEMP_SENSOR_CHANNEL  0x70       /* internal temperature sensor */

/* Variables global to eliminate parameter passing and visibility in emulator */
unsigned int   Kref [3];                 /* 24 bit unsigned integer */
unsigned int   KrefLo @ Kref;
unsigned int   KrefMid @ Kref+1;
unsigned int   KrefHi @ Kref+2;
unsigned int   Krefexp;
unsigned int   Kbg [3];                  /* 24 bit unsigned integer */
unsigned int   KbgLo @ Kbg;
unsigned int   KbgMid @ Kbg+1;
unsigned int   KbgHi @ Kbg+2;
unsigned int   Kbgexp;
unsigned long  Nbg;
unsigned int   NbgLo @ Nbg;
unsigned int   NbgHi @ Nbg+1;
unsigned long  Noffset;
unsigned int   NoffsetLo @ Noffset;
unsigned int   NoffsetHi @ Noffset + 1;
unsigned long  Nbattery;
unsigned int   NbatteryLo @ Nbattery;
unsigned int   NbatteryHi @ Nbattery + 1;
unsigned long  Vbattery;
unsigned int   VbatteryLo @ Vbattery;
unsigned int   VbatteryHi @ Vbattery + 1;
unsigned long  ChargeState;

#include "math32.c"
#include "cmp-ref.c"
#include "timer.c"

/*************************************************************************
*       RunA2DConv
*              runs a conversion on the currently selected AD channel.
*
*              Input Variables:
*                     None
*              Output Variables:
*                     Returns ADCOUNT value.
*************************************************************************/
unsigned long RunA2DConv ()
{
    unsigned long  adcounts @ ADCAPL;

    ADCON1 &= 0x0F;
    ADCON1 |= 0xC0;          /* select current constant (27uA). */
    PIR1.ADCIF = 0;
    SLPCON.REFOFF = 0;
    SLPCON.ADOFF = 0;        /* enable the AD module */
    ADCON0.ADRST = 1;        /* Stop timer and fully dischage ramp capacitor */
    Delay_10xUs_4MHz (50);   /* delay 500 us */
    ADTMRH = 0;
    ADTMRL = 0;              /* clear the conversion clock */
    ADCON0.ADRST = 0;        /* start conversion */
    while (!(PIR1.ADCIF));   /* wait for conversion to complete */

    return (adcounts);
}

/*************************************************************************
*       Calibrate_AD
*              Runs an AD conversion on the High and Low references and
*              calculates the offset value.  This is necessary to account
*              for capacitor dialectric absorbtion and comparator offset.
*              For more information, see AN624.
*
```

```
*               Input Variables:
*                       Kref        (global) slope reference value from
*                                   calibration table.
*               Output Variables:
*                       Noffset (global)  AD counts to subtract to account
*                                   for capacitor dielectric absorbtion and
*                                   comparator offset
*               Return Value:
*                       None
*******************************************************************************/
void Calibrate_AD ()
{
    unsigned long  difference;
    unsigned int   difflo @ difference;
    unsigned int   diffhi @ difference+1;
    unsigned long  Nrefhi;
    unsigned long  Nreflo;
    unsigned int   i;
    unsigned int   round;

    reg1 [0] = 0;
    reg1 [1] = 0;
    reg1 [2] = 0;
    reg1 [3] = 0;
    for (i = 0; i < 16; i++)
    {
       ADCON0 &= 0x0F;
       ADCON0 |= 0x50;  /* Select SREFHI */
       Nrefhi = RunA2DConv ();

       ADCON0 &= 0x0F;
       ADCON0 |= 0x60;  /* Select SREFLO */
       Nreflo = RunA2DConv ();

       difference = (Nrefhi - Nreflo);
       reg2 [0] = 0;
       reg2 [1] = 0;
       reg2 [2] = diffhi;
       reg2 [3] = difflo;   /* binary point all the way to the right */
       add32 ();
       reg1 [0] = reg3 [0];
       reg1 [1] = reg3 [1];
       reg1 [2] = reg3 [2];
       reg1 [3] = reg3 [3];
    }

    round = reg3 [3]; /* save off the highest order bit that will be lost*/
    for (i = 0; i < 4; i++)
    {
#asm
       bcf       STATUS,RP0       /* assembly is necessary here.  Using the */
       bcf       STATUS,C         /* C shift operator (>>) doesn't work     */
       rrf       reg1,1           /* here because it clears the carry bit   */
       rrf       reg1+1,1         /* between shifts.                        */
       rrf       reg1+2,1
       rrf       reg1+3,1
#endasm
    }
    diffhi = reg3 [2];
    difflo = reg3 [3];
    if (round & 0x08)              /* complete rounding operation rather than truncation */
       ++ difference;

//    difference *= Kref;
    reg2 [0] = 0;
    reg2 [1] = KrefHi;
```


© 1997 Microchip Technology Inc.

```
    reg2 [2] = KrefMid;
    reg2 [3] = KrefLo;
    Shift_R2_Left ();  /* align the binary point just ahead of reg2 [1] */

    mult32 ();         /* binary point now between req3[4] and reg3[5] */
    difflo = reg3 [4];
    diffhi = reg3 [3];

       if (difference > Nreflo)  /* check to see Noffset would be negative */
           Noffset = 0;
       else
           Noffset = Nreflo - difference;
}


/*****************************************************************************
*       ADC_Counts
*             Do a conversion on the specified AD channel.  Channel is
*             measured 16 times and averaged.
*
*             Input Variables:
*                    channel  (Paramater) AD MUX channel - ADCON0(7:4) per table 8-1.
*                    Noffset  (Global)    16 bit unsigned int
*             Output Variables:
*                    None
*             Return Value
*                    Vbattery 16 bit fixed point
*****************************************************************************/
unsigned long ADC_Counts (unsigned int channel)
{
    unsigned long  Ncounts;
    unsigned int   NcountsHi @ Ncounts+1;
    unsigned int   NcountsLo @ Ncounts;
    unsigned int   i;
    unsigned int   round;

    for (i = 0; i < 4; i++)
       reg1[i] = 0;

    ADCON0 &= 0x0F;
    ADCON0 |= (channel & 0xF0);
    for (i = 0; i < 16; i++)
    {
       Ncounts = RunA2DConv ();
       Ncounts -= (long) Noffset;
       reg2[0] = 0;
       reg2[1] = 0;
       reg2[2] = NcountsHi;
       reg2[3] = NcountsLo;
       add32 ();
       reg1[0] = reg3[0];
       reg1[1] = reg3[1];
       reg1[2] = reg3[2];
       reg1[3] = reg3[3];
    }

    round = reg1[3];
    round &= 0x08;  /* save off the highest order bit that will be lost */
    for (i = 0; i < 4; i++)
    {
#asm
        bcf STATUS, RP0
        bcf STATUS, C
        rrf reg1, 1
        rrf reg1+1, 1
        rrf reg1+2, 1
```

```
        rrf reg1+3, 1
#endasm
    }
    NcountsHi = reg1[2];
    NcountsLo = reg1[3];
    if (round)
       Ncounts++;
    return (Ncounts);
}


/*************************************************************************
*       AN624Eq5
*              Takes the conversions previously done on the battery and the
*              Bandgap, along with the calibration data in memory, and
*              performs AN 624 Equation 5 to convert the battery counts
*              back to the original voltage.
*
*              An 624 Equation 5:
*                     V = (Nin - Noffset) / (Nbg - Noffset) * Kbg
*                     (Subtraction of Noffset is performed in ADC_Counts).
*              Actual Calculation performed here:
*                     V = AvgNin * Kbg / AvgNbg
*              This answer gives the voltage at the ADC, however the circuit
*              uses a resistive divider to take the ~12V at the battery down
*              to the ~1.2V for the ADC.  We need to multiply by the ratio
*              of the resistances to get actual battery voltage.
*                     Vbattery = V * (1Mohm + 68Kohm)/68Kohm
*                              = V * 15.70588  (Nominal values)
*              For best accuracy, use measured values on the resistors
*                              = V * 15.95981  (Actual values: 1072180 / 67180)
*
*              Input Variables:
*                     Nbattery unsigned int (16 bits)
*                     Nbg      unsigned int (16 bits)
*                     Kbg      fixed point  (24 bits, 16 behind decimal)
*              Output Variables:
*                     None
*              Return Value
*                     Vbattery fixed point (16 bits, 8 behind decimal)
*************************************************************************/
unsigned long AN624Eq5 ()
{
//    Vin = Kbg * Nbattery;
    reg1 [0] = 0;
    reg1 [1] = 0;
    reg1 [2] = NbatteryHi;
    reg1 [3] = NbatteryLo;

    reg2 [0] = 0;
    reg2 [1] = KbgHi;
    reg2 [2] = KbgMid;
    reg2 [3] = KbgLo;
    Shift_R2_Left ();  // align the decimal point
    mult32 ();         // answer in Reg3

//    Vin /= Nbg;  /* AN624 Equation 5.  (Subtractions previously done) */

    reg2 [0] = 0;
    reg2 [1] = 0;
    reg2 [2] = NbgHi;
    reg2 [3] = NbgLo;

    reg3 [0] = reg3 [3];
    reg3 [1] = reg3 [4];
    reg3 [2] = reg3 [5];
```


© 1997 Microchip Technology Inc.

```
    reg3 [3] = reg3 [6];

    div32 ();

//    Vbattery = Vin * 1062000/62000;
    reg2 [0] = 0;     // 15.70588 = 1068000/68000  (Nominal values)
    reg2 [1] = 15;    // 15.9598  = 1072180/67180  (measured values)
    reg2 [2] = 180;   // decimal portion (.9598 * 256)
    reg2 [3] = 181;

    mult32 ();

    VbatteryHi = reg3 [3];
    VbatteryLo = reg3 [4];

    return (Vbattery);
}


/*****************************************************************************
*       GetCalData
*              retrieves the calibration parameters, and converts them to
*              fixed point format.
*
*              Input Variables:
*                     Name       Description of what the variable is used for
*              Output Variables:
*                     Kref       fixed point (24 bits, 23 behind decimal point)
*                     Kgb        fixed point (24 bits, 23 behind decimal point)
*****************************************************************************/
void  GetCalData ()
{

#asm
       bsf          PCLATH,3  ; select page 1
       bcf          STATUS,RP0
       call         0x07c0
       movwf        Krefexp
       call         0x07c1
       IORLW        0x80      ; ignore sign bit, force implied bit
       movwf        KrefHi
       call         0x07c2
       movwf        KrefMid
       call         0x07c3
       movwf        KrefLo

       call         0x07c4
       movwf        Kbgexp
       call         0x07c5
       IORLW        0x80      ; ignore sign bit, force implied bit
       movwf        KbgHi
       call         0x07c6
       movwf        KbgMid
       call         0x07c7
       movwf        KbgLo

       bcf          PCLATH, 3
#endasm
       for (; Krefexp < 0x7f; Krefexp ++)
       {    //   Kref >>= 1;
#asm
       bcf          STATUS,RP0
       bcf          STATUS,C
       rrf          KrefHi,F
       rrf          KrefMid,F
       rrf          KrefLo,F
```

```
#endasm
        }

        for (; Kbgexp < 0x7f; Kbgexp ++)
        {     // Kbg >>= 1;
#asm
        bcf         STATUS,RP0
        bcf         STATUS,C
        rrf         KbgHi,F
        rrf         KbgMid,F
        rrf         KbgLo,F
#endasm
        }
        return;
}


/*************************************************************************
*       Select New Charge
*             Takes the current charge state, battery voltage and time in
*             the current state and comes up with the new state.  If a new
*             state is entered, the timer is reset.
*
*             Input Variables:
*                   BatteryVoltage      (Parameter)       Current battery voltage
*                   ChargeState         (Global)          Current Charge State
*                                                         (Hi, Low or No current)
*             Output Variables:
*                   ChargeState         (Global)          New Charge State
*************************************************************************/
void select_new_charge (unsigned long BatteryVoltage)
{
    unsigned long  ChargeTime;

    ChargeTime = ChargeMinutes ();  /* how long have we been in the current charge state */
    if (ChargeState == HIGH_CURRENT)
    {
       if ((BatteryVoltage >= V_LIMIT_HIGH)
        || (ChargeTime >= HIGH_CHARGE_TIME_LIMIT))
       {
           ChargeState = NO_CURRENT;
           ResetTimer ();
       }
    }
    else if (ChargeState == LOW_CURRENT)
    {
       if ((BatteryVoltage >= V_LIMIT_HIGH)
        || (ChargeTime >= LOW_CHARGE_TIME_LIMIT))
       {
           ChargeState = NO_CURRENT;
           ResetTimer ();
       }
    }
    else if ((BatteryVoltage <= V_LIMIT_LOW)
         && (ChargeTime >= CHARGE_REST_TIME))
    {
       ChargeState = LOW_CURRENT;
       ResetTimer ();
    }
}


/*************************************************************************
*       Setup WDT
*             Sets up the WatchDog Timer for a 64:1 prescale.  This will
*             wake the part up 1.15 seconds after it goes to sleep.
```


© 1997 Microchip Technology Inc.

```
*              with interrupts disabled, processing resumes immediately
*              after the sleep instruction.  Ref PIC14C000 Data sheet,
*              section 10.7 (1996/1997 databook).
*
*              Input Variables:
*                     None
*              Output Variables:
*                     None
*************************************************************************/
void Setup_WDT ()
{
    OPTION = 0xCE;  /* Prescaler on WDT, 64:1 prescale */
    INTCON = 0x00;  /* all interrupts disabled */
    CLRWDT ();
}


/*************************************************************************
* Main Loop:  starts of by reading battery voltage and determining
* type of charge needed (HIGH CURRENT, LOW CURRENT, NO CURRENT).
* Then every second it takes a voltage reading on the battery.  If it's
* above the highest limit, it turns off the charge.  If it's drained
* down below the lower limit, it turns on the trickle charge.  Only
* way it can get set to fast charge is on startup.
*
* The processor is put to sleep for remainder of charging cycle (about 1S).
* WDT is setup to wake up the processor for the next cycle.
*
* state transition diagram:  HIGH -> OFF <---> LOW
* HIGH transitions to OFF when V_LIMIT_HIGH is exceeded.
* OFF transitions to LOW when battery voltage has drained below V_LIMIT_LOW
* LOW transitions to OFF when V_LIMIT_HIGH is exceeded.
*
*              Input Variables:
*                     None
*              Output Variables:
*                     None
*              Returned Value
*                     None
*************************************************************************/
void main ()
{
    unsigned long   BVoltage;  /* battery voltage    Fixed point (16:8) */
    unsigned int i;

    TRISD = 0x30;   /* AN 4 and 5 inputs, rest of port D all outputs */
    StopCharge ();  /* just in case it was running previously */
    GetCalData ();  /* get the Kref & Kbg values from cal data memory */

    Calibrate_AD ();
    Nbattery = ADC_Counts (BATTERY_CHANNEL);
    Nbg      = ADC_Counts (BAND_GAP_CHANNEL);
    BVoltage = AN624Eq5 ();

    if (BVoltage < V_LIMIT_LOW)
       ChargeState = HIGH_CURRENT;
    else if (BVoltage > V_LIMIT_HIGH)
       ChargeState = NO_CURRENT;
    else
       ChargeState = LOW_CURRENT;
    ResetTimer ();
    Setup_WDT ();
    while (1)  /* loop forever */
    {
       ChargeCurrent (ChargeState);
       SLEEP ();
```

```
        BumpTimer ();
        Calibrate_AD ();    /* Calculates Noffset */
        StopCharge ();
        Nbattery = ADC_Counts (BATTERY_CHANNEL);
        Nbg      = ADC_Counts (BAND_GAP_CHANNEL);
        ChargeCurrent (ChargeState);        /* turn charger back on while we */
        BVoltage = AN624Eq5 ();               /* crunch the numbers */
        select_new_charge (BVoltage);
    }
}
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: CMP-REF.C

```
/* ****************************************************************  */
/* Comparator - Reference utilities                                  */
/*                                                                   */
/* Functions:                                                        */
/*    StopCharge - turn off the comparator and force RD2 low         */
/*    ChargeCurrent - Calculates the appropriate Vref value and      */
/*          Sets up Voltage Reference and Comparator B.              */
/* ****************************************************************  */
/* these two arrays form a look up table for the Programmable voltage */
/* reference.  The top voltage of the range is in the first table,    */
/* and the corresponding coarse bits for the programmable reference   */
/* are in the second table.  The lookup then subtracts off the bottom */
/* of the range and divides by the set size to get the fine bits.     */
/* ****************************************************************  */
const unsigned long  TopVoltage [32] = {
       13107,  /* .1500 - .2000 */
       16384,  /* .2000 - .2500 */
       19660,  /* .2500 - .3000 */
       22937,  /* .3000 - .3500 */
       26214,  /* .3500 - .4000 */
       29491,  /* .4000 - .4500 */
       29497,  /* .4500 - .4550 */
       29818,  /* .4550 - .4600 */
       30146,  /* .4600 - .4650 */
       30474,  /* .4650 - .4700 */
       30801,  /* .4700 - .4750 */
       31129,  /* .4750 - .4800 */
       31457,  /* .4800 - .4850 */
       32112,  /* .4850 - .4900 */
       32440,  /* .4900 - .4950 */
       32768,  /* .4950 - .5000 */
       33095,  /* .5000 - .5050 */
       33423,  /* .5050 - .5100 */
       33751,  /* .5100 - .5150 */
       34078,  /* .5150 - .5200 */
       34406,  /* .5200 - .5250 */
       34734,  /* .5250 - .5300 */
       35061,  /* .5300 - .5350 */
       35389,  /* .5350 - .5400 */
       35717,  /* .5400 - .5450 */
       36044,  /* .5450 - .5500 */
       39321,  /* .5500 - .6000 */
       42598,  /* .6000 - .6500 */
       45875,  /* .6500 - .7000 */
       49152,  /* .7000 - .7500 */
       52428,  /* .7500 - .8000 */
       55705}; /* .8000 - .8500 */

const unsigned int   Coarse [32] =
       {0xf8, /* .1500 - .2000 */
       0xf0,  /* .2000 - .2500 */
       0xe8,  /* .2500 - .3000 */
       0xe0,  /* .3000 - .3500 */
       0xd8,  /* .3500 - .4000 */
       0xd0,  /* .4000 - .4500 */
       0xc8,  /* .4500 - .4550 */
       0xc0,  /* .4550 - .4600 */
       0xb8,  /* .4600 - .4650 */
       0xb0,  /* .4650 - .4700 */
       0xa8,  /* .4700 - .4750 */
```


© 1997 Microchip Technology Inc.

```
        0xa0,  /* .4750 - .4800 */
        0x98,  /* .4800 - .4850 */
        0x90,  /* .4850 - .4900 */
        0x88,  /* .4900 - .4950 */
        0x80,  /* .4950 - .5000 */
        0x00,  /* .5000 - .5050 */
        0x08,  /* .5050 - .5100 */
        0x10,  /* .5100 - .5150 */
        0x18,  /* .5150 - .5200 */
        0x20,  /* .5200 - .5250 */
        0x28,  /* .5250 - .5300 */
        0x30,  /* .5300 - .5350 */
        0x38,  /* .5350 - .5400 */
        0x40,  /* .5400 - .5450 */
        0x48,  /* .5450 - .5500 */
        0x50,  /* .5500 - .6000 */
        0x58,  /* .6000 - .6500 */
        0x60,  /* .6500 - .7000 */
        0x68,  /* .7000 - .7500 */
        0x70,  /* .7500 - .8000 */
        0x78}; /* .8000 - .8500 */

#define  GREEN_ON  PORTC.2 = 0       /* Macro to turn on Green  LED */
#define  GREEN_OFF PORTC.2 = 1       /* Macro to turn off Green LED */
#define  RED_ON    PORTC.3 = 0       /* Macro to turn on Red    LED */
#define  RED_OFF   PORTC.3 = 1       /* Macro to turn off Red   LED */

/*************************************************************************
*       StopCharge
*              Disables the charge comparator and turns off the indicators
*
*              Input Variables:
*                     None
*              Output Variables:
*                     None
*              Return Value
*                     None
*************************************************************************/
void StopCharge ()
{
    CHGCON.CMBOE = 0;  /* Disconnect RD2 from Comparator */
    PORTD.2 = 0;       /* Set RD2 LOW (turn off FET)     */
    RED_OFF;
    GREEN_OFF;
}

/*************************************************************************
*       StartCharge
*              Sets up the comparator and programmable voltage reference
*
*              Input Variables:
*                     PrefValue - Value for Programmable Voltage Ref
*                     See PIC14C000 DataSheet tables 9-1 and 9-2.
*              Output Variables:
*                     None
*              Return Value
*                     None
*************************************************************************/
void StartCharge (PrefValue)
     unsigned int  PrefValue;
{
    SLPCON.CMOFF = 0;      /* enable comparators */
    SLPCON.REFOFF = 0;     /* enable power control references. */
    SLPCON.LSOFF = 0;      /* enable level shift network  */

    CHGCON.CMBOE = 1;      /* comparator B output on RD2     */
```


© 1997 Microchip Technology Inc.

```
    CHGCON.CPOLB = 1;       /* comparator B output inverted.  */
    PREFB = PrefValue;
}

/*************************************************************************
*  ChargeCurrent
*       Sets up the constant current charge on comparator B, based on
*       the charge state.  Also turns on the LED charge indicators.
*       (Red is Fast charge, Green is slow charge)
*
*              Input Variables:
*                     charge_current   (parameter)
*                     charge rate      (#define, fixed point (16:8)
*                     Sense Resistor   (#define, fixed point (16:16)
*              Output Variables:
*                     PREFB Programmable Voltage Reference B
*              Return Value
*                     None
*
*************************************************************************/
void ChargeCurrent (ChargeRate)
    unsigned long  ChargeRate;
{
    unsigned long  ControlV;
    unsigned int ControlVHi @ ControlV+1;
    unsigned int ControlVLo @ ControlV;
    unsigned long step;
    unsigned long templong;
    unsigned int fine;   /* fine adjust bits of PREFB */
    unsigned int coarse;
    unsigned int i,j;

    TRISD = 0x30;           /* Set AN4 and AN5 for input */

    RED_OFF;
    GREEN_OFF;
    TRISC = 0x00;          /* RC for output */


    reg1 [0] = 0;
    reg1 [1] = 0;
    reg1 [2] = (ChargeRate & 0xFF00) >> 8;
    reg1 [3] = ChargeRate & 0x00FF;       /* decimal point before here */

    reg2 [0] = 0;
    reg2 [1] = 0;
    reg2 [2] = (SENSE_RESISTOR & 0xFF00) >> 8; /* decimal point before here */
    reg2 [3] = SENSE_RESISTOR & 0x00FF;
    mult32 ();
    ControlVLo = reg3 [6];   /* keep most significant bits */
    ControlVHi = reg3 [5];
    ControlV += LEVEL_SHIFT;

    for (i = 0; i < 32; i++)       /* now need to convert ControlV to PREFB value */
    {
       templong = TopVoltage [i];
       if (ControlV < templong)
       {
           if ((i < 6) || (i > 25))
              step = 409;
           else
              step = 41;
           j = i-1;
           templong = TopVoltage [j];
           ControlV -= templong;
           fine = ControlV / step;
```

```
            coarse = Coarse [i];
          break;
       }
    }

    if (ChargeRate == HIGH_CURRENT)
    {
       RED_ON;             /* turn on red LED */
       StartCharge (coarse | fine);
    }
    else if (ChargeRate == LOW_CURRENT)
    {
       GREEN_ON;           /* Turn on green LED */
       StartCharge (coarse | fine);
    }
    else
       StopCharge ();
}
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C: TIMER.C

```
/************************************************************************
*  Filename: timer.c
************************************************************************
*       Author:         Dan Butler
*       Company:        Microchip Technology
*       Revision:       Rev 1.0
*       Date:           29 January 1997
*       Compiler:       MPLAB-C rev 1.10
************************************************************************
*        Include files:
*              none
*
************************************************************************
*
*  Implements a timer operation:
*       ResetTimer
*       BumpTimer
*       ChargeMinutes
*       ChargeSeconds
*
*       Clock Frequency     4 MHz Internal RC
*       Configuration Bit Settings   WDT on
*       Program and Data Memory Usage
*
************************************************************************
*  What's Changed
*
*  Date              Description of Change
*
*
*
************************************************************************/
unsigned int   seconds;
unsigned int   correction;
unsigned long  minutes;

/************************************************************************
*       ResetTimer
*              Sets the timer counters all back to zero.
*
*              Input Variables:
*                    None
*              Output Variables:
*                    None
************************************************************************/
void  ResetTimer ()
{
    seconds = 0;
    minutes = 0;
    correction = 0;
}


/************************************************************************
*       BumpTimer
*              bumps the timer by 1 second.  Since the WDT timeout period
*              is actually 1.15 seconds, plus another .1 per cycle through
*              the code for a total of 1.25 S/cycle, we add an extra
*              second for each 4th time called.  Accuracy at room
*              temperature has been measured at better than 6 seconds per
```

```
*               hour (0.17%).
*
*               Input Variables:
*                       None
*               Output Variables:
*                       None
*******************************************************************************/
void  BumpTimer ()
{
    if (++seconds == 60)
    {
        ++ minutes;
       seconds = 0;
    }

    if (++correction == 4)
    {
       if (++seconds == 60)
       {
            ++ minutes;
           seconds = 0;
       }
       correction = 0;
    }
}



/*******************************************************************************
*       ChargeMinutes
*               returns the number of minutes the charge cycle has been in
*               progress.
*
*               Input Variables:
*                       None
*               Output Variables:
*                       None
*******************************************************************************/
unsigned long  ChargeMinutes ()
{
    return (minutes);
}



/*******************************************************************************
*       ChargeSeconds
*               returns the number of seconds portion of the charge timer
*
*               Input Variables:
*                       None
*               Output Variables:
*                       None
*******************************************************************************/
unsigned int  ChargeSeconds ()
{
    return (seconds);
}
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX D: MATH32.C

```
/*****************************************************************************
*  Filename: MATH32.C
*****************************************************************************
*       Author:                         Dan Butler
*              Company:                 Microchip Technology
*              Revision:        Rev 1.0
*       Date:                           29 January 1997
*              Compiler:        MPLAB-C rev 1.10
*****************************************************************************
*       Include files:
*       Math.h          Version 1.00
*
*****************************************************************************
*
* ASMD & Shift operations on 32 bit unsigned integers
*
*       Clock Frequency      4 MHz Internal RC
*              Configuration Bit Settings    WDT on
*       Program and Data Memory Usage
*
*****************************************************************************
*  What's Changed
*
*  Date               Description of Change
*
*
*
*****************************************************************************/
#include <14000.h>
#include <math.h>
unsigned int reg1 [4];        //math routine registers - 32 bits
unsigned int reg2 [4];        //32 bits normally, but need 64 for the div.
unsigned int reg3 [8];        //64 bits - used for multiply routine
unsigned int Quotient [4];
unsigned int carry;           //flag register for math routine
unsigned int sign;            // 1 - positive or zero, 0 - negative
unsigned long longtemp;
unsigned long X, Y;
unsigned int i,j;

/*****************************************************************************
*       add32
*              32 bit unsigned addition reg3 = reg1 + reg2
*
*              Input Variables:
*                     reg1 - 32 bit unsigned integer
*                     reg2 - 32 bit unsigned integer
*              Output Variables:
*                     reg3 - 32 bit unsigned integer
*                     carry - overflow
*****************************************************************************/
void add32 ()
{
    carry = 0;
    for (i = 3; i != 0xFF; i--)
    {
       X = reg1 [i];
       Y = reg2 [i];
       longtemp = X + Y + carry;
       reg3 [i] = (unsigned int) longtemp;
```

```
        carry = longtemp >> 8;
    }
}

/*************************************************************************
*       sub32
*
*               32bit unsigned subtraction:  reg1 = reg3 - reg2
*
*               Input Variables:
*                       reg3 - 32 bit unsigned integer
*                       reg2 - 32 bit unsigned integer
*               Output Variables:
*                       reg1 - 32 bit unsigned integer
*                       sign -  1: positive or zero.  0: negative
*************************************************************************/
void sub32 ()
{
#asm
        movf        reg3,w    ; copy Reg3 to Reg1
        movwf       reg1
        movf        reg3+1,w
        movwf       reg1+1
        movf        reg3+2,w
        movwf       reg1+2
        movf        reg3+3,w
        movwf       reg1+3

; Reg + 3
        movf        reg2+3,w    ;  subtract low byte
        subwf       reg1+3,1
; Reg + 2
        movf        reg2+2,w    ; move borrow bit to W reg
        btfss       STATUS,
        incfsz      reg2+2,w
        subwf       reg1+2,1

; Reg + 1
        movf        reg2+1,w
        btfss       STATUS,
        incfsz      reg2+1,w
        subwf       reg1+1,1

; Reg + 0
        movf        reg2,w
        btfss       STATUS,
        incfsz      reg2,w
        subwf       reg1,1

        movf        STATUS,w    ; move borrow bit to W reg
        andlw       0x01          ; get rid of the rest
        movwf       sign,1
#endasm
}

/*************************************************************************
*       mult32
*               32 bit unsigned multiplication: reg3 = reg1 * reg2
*
*               Input Variables:
*                       reg1 - 32 bit unsigned integer
*                       reg2 - 32 bit unsigned integer
*               Output Variables:
*                       reg3 - 64 bit unsigned integer
```


© 1997 Microchip Technology Inc.

```
*************************************************************************/
void mult32 ()
{
    for (i = 0; i < 8; i++)
       reg3 [i] = 0;

    for (i = 3; i != 0xFF; i--)
    {
        carry = 0;
       for (j = 3; j != 0xFF; j--)
       {
           X = reg1 [i];
           Y = reg2 [j];
           longtemp = X * Y;
           longtemp += reg3 [i + j + 1];
           longtemp += carry;
           reg3 [i + j + 1] = (unsigned int) longtemp;
           carry = longtemp >> 8;
       }
       reg3 [i] = carry;
    }
}


/*************************************************************************
*       Shift_R2_Left
*
*              Shifts all 32 bits of reg2 left one position:
*                     reg2 <<= 1;
*
*              Input Variables:
*                     reg2 - 32 bit unsigned integer
*              Output Variables:
*                     reg2 - 32 bit unsigned integer
*************************************************************************/
void Shift_R2_Left ()
{
#asm
   bcf       STATUS,C
   rlf       reg2+3,1
   rlf       reg2+2,1
   rlf       reg2+1,1
   rlf       reg2,1
#endasm
}


/*************************************************************************
*       Shift_R2_Left
*
*              Shifts all 32 bits of Quotient left one position:
*                     Quotient <<= 1;
*
*              Input Variables:
*                     Quotient - 32 bit unsigned integer
*              Output Variables:
*                     Quotient - 32 bit unsigned integer
*************************************************************************/
void Shift_Q_Left ()
{
#asm
   bcf       STATUS,C
   rlf       Quotient+3,1
   rlf       Quotient+2,1
   rlf       Quotient+1,1
   rlf       Quotient,1
```


© 1997 Microchip Technology Inc.

```
#endasm
}


/*************************************************************************
*       Shift_R2_Right
*
*               Shifts all 32 bits of reg2 right one position:
*                       reg2 >>= 1;
*
*               Input Variables:
*                       reg2 - 32 bit unsigned integer
*               Output Variables:
*                       reg2 - 32 bit unsigned integer
*************************************************************************/
void Shift_R2_Right ()
{
#asm
   bcf       STATUS,C
   rrf       reg2,1
   rrf       reg2+1,1
   rrf       reg2+2,1
   rrf       reg2+3,1
#endasm
}


/*************************************************************************
*       div32
*
*               32 bit unsigned division
*                       reg1 = reg3 / reg2
*
*               Input Variables:
*                       reg3 - 32 bit unsigned integer
*                       reg2 - 32 bit unsigned integer
*               Output Variables:
*                       reg1 - 32 bit unsigned integer
*************************************************************************/
void div32 ()
{
    i = 0;
    while (!(reg2[0] & 0x80))
    {
       Shift_R2_Left ();
       ++i;
    }

    Quotient [0] = 0;
    Quotient [1] = 0;
    Quotient [2] = 0;
    Quotient [3] = 0;

    for (j = 0; j <= i; j++)
    {
       Shift_Q_Left ();
       sub32 ();
       if (sign)  // was the result positive?
       {
           reg3 [0] = reg1 [0];
           reg3 [1] = reg1 [1];
           reg3 [2] = reg1 [2];
           reg3 [3] = reg1 [3];
           Quotient [3] |= 0x01;
       }
       Shift_R2_Right();
```


© 1997 Microchip Technology Inc.

```
    }
    reg1 [0] = Quotient [0];
    reg1 [1] = Quotient [1];
    reg1 [2] = Quotient [2];
    reg1 [3] = Quotient [3];
}
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN643

Adaptive Differential Pulse Code Modulation using PICmicro™ Microcontrollers

Author: *Rodger Richey*
Microchip Technology Inc.

INTRODUCTION

In the past, adding speech recording and playback capability to a product meant using a digital signal processor or a specialized audio chip. Now, using a simplified Adaptive Differential Pulse Code Modulation (ADPCM) algorithm, these audio capabilities can be added to any PICmicro device. This application note will cover the ADPCM compression and decompression algorithms, performance comparison of all PICmicro devices, and an application using a PIC16C72 microcontroller.

DEFINITION OF TERMS

step size - value of the step used for quantization of analog signals and inverse quantization of a number of steps.
quantization - the digital form of an analog input signal is represented by a finite number of steps.
adaptive quantization - the step size of a quantizer is dramatically changed with time in order to adapt to a changing input signal.
inverse quantizer - a finite number of steps is converted into a digital representation of an analog signal.

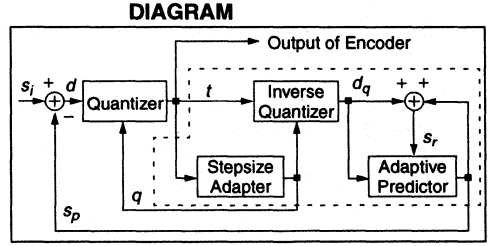
THEORY OF OPERATION

The ADPCM algorithm takes advantage of the high correlation between consecutive speech samples, which enables future sample values to be predicted. Instead of encoding the speech sample, ADPCM encodes the difference between a predicted sample and the speech sample. This method provides more efficient compression with a reduction in the number of bits per sample, yet preserves the overall quality of the speech signal. The implementation of the ADPCM algorithm provided in this application note is based on the Interactive Multimedia Association's (IMA) Recommended Practices for Enhancing Digital Audio Compatibility in Multimedia Systems revision 3.00. The ITU (formerly CCITT) G.721 ADPCM algorithm is well known and has been implemented on many digital signal processors such as the TMS320 family from Texas Instruments and the ADSP-2100 family from Analog Devices. ITU G.721 uses floating point arithmetic and logarithmic functions which are not easily implemented in the 8-bit microcontroller world. The IMA Reference algorithm significantly reduces the mathematical complexity of ITU G.721 by simplifying many of the operations and using table lookups where appropriate.

COMPRESSION

The input, s_i, to the encoder routine must be 16-bit two's complement speech data. The range of allowable values for s_i is 32767 to -32768. Figure 1 shows a block diagram for ADPCM compression and Appendix A has a listing of the function `ADPCMEncoder()`. The predicted sample, s_p, and the quantizer step size index are saved in a structure for the next iteration of the encoder. Initially, the quantizer step size index and the predicted sample (s_p) are set to zero. The encoder function takes a 16-bit two's complement speech sample and returns an 8-bit number containing the 4-bit sign-magnitude ADPCM code.

FIGURE 1: ADPCM ENCODER BLOCK DIAGRAM



The predicted sample, s_p, is subtracted from the linear input sample, s_i, to produce a difference, d. Adaptive quantization is performed on the difference, resulting in the 4-bit ADPCM value, t. The encoder and decoder both update their internal variables based on this ADPCM value. A full decoder is actually embedded within the encoder. This ensures that the encoder and decoder are synchronized without the need to send any additional data. The embedded decoder is shown within the dotted lines of Figure 1. The embedded decoder uses the ADPCM value to update the inverse quantizer, which produces a dequantized version, d_q, of the difference, d. The implementation of the ADPCM algorithm presented in this application note uses a fixed predictor instead of an adaptive predictor which reduces the amount of data memory and instruction cycles required.

© 1997 Microchip Technology Inc.

The adaptive predictor of ITU G.721 adjusts according to the value of each input sample using a weighted average of the last six dequantized difference values and the last two predicted values. So at this point, the dequantized difference, d_q, is added to the predicted sample, s_p, to produce a new predicted sample, s_r. Finally the new predicted sample, s_r, is saved into s_p.

The following (Table 1) is a step-by-step description of the **`ADPCMEncoder()`** function from Appendix A.

TABLE 1: `ADPCMEncoder()` STEP-BY-STEP FUNCTIONS

1. ADPCMEncoder takes a 16-bit signed number (speech sample, 32767 to -32768) and returns an 8-bit number containing the 4-bit ADPCM code (0-15)	`char ADPCMEncoder(long signed sample)`
2. Restore the previous values of predicted sample (s_p) and the quantizer step size index	`predsample = state.prevsample;` `index = state.previndex;`
3. Find the quantizer step size (q) from a table lookup using the quantizer step size index	`step = StepSizeTable[index];`
4. Compute the difference (d) between the actual sample (s_i) and the predicted sample (s_p)	`diff = sample - predsample;`

5. Set the sign bit of the ADPCM code (i) if necessary and find the absolute value of difference (d)	`if(diff >= 0)`
	`  code = 0;`
	`else`
	`{`
	`  code = 8;`
	`  diff = - diff;`
	`}`

6. Save quantizer step size (q) in a temporary variable	`tempstep = step;`

7. Quantize the difference (d) into the ADPCM code (i) using the quantizer step size (q)	`if(diff >= tempstep)`
	`{`
	`  code |= 4;`
	`  diff -= tempstep;`
	`}`
	`tempstep >>= 1;`
	`if(diff >= tempstep)`
	`{`
	`  code |= 2;`
	`  diff -= tempstep;`
	`}`
	`tempstep >>= 1;`
	`if(diff >= tempstep)`
	`  code |= 1;`

8. Inverse quantize the ADPCM code (i) into a predicted difference (d_q) using the quantizer step size (q)	`diffq = step >> 3;`
	`if(code & 4)`
	`  diffq += step;`
	`if(code & 2)`
	`  diffq += step >> 1;`
	`if(code & 1)`
	`  diffq += step >> 2;`

© 1997 Microchip Technology Inc.

TABLE 1: `ADPCMEncoder()` STEP-BY-STEP FUNCTIONS (CONTINUED)

9. Fixed predictor computes new predicted sample (s_r) by adding the old predicted sample (s_p) to the predicted difference (d_q)	`if(code & 8)`
	`  predsample -= diffq;`
	`else`
	`  predsample += diffq;`

10. Check for overflow of the new predicted sample (s_r). s_n which is a signed 16-bit sample, must be in the range of 32767 to -32768	`if(predsample > 32767)`
	`  predsample = 32767;`
	`else if(predsample < -32768)`
	`  predsample = -32768;`

11. Find the new quantizer step size index (q) by adding the previous index and a table lookup using the ADPCM code (*t*)	`index += IndexTable[code];`

12. Check for overflow of the new quantizer step size index	`if(index < 0)`
	`  index = 0;`
	`if(index > 88)`
	`  index = 88;`

13. Save the new predicted sample (s_r) and quantizer step size index for next iteration	`state.prevsample = predsample;` `state.previndex = index;`
14. Return the ADPCM code (*t*)	`return (code & 0x0f);`

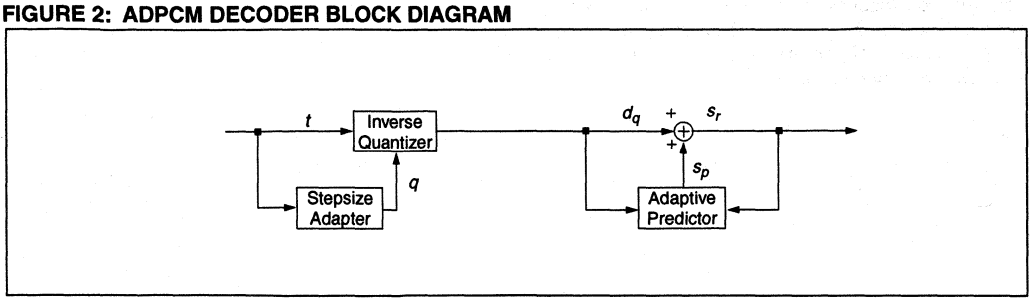
This function requires five 16-bit variables and two 8-bit variables. Some optimizations can be made to the code in Appendix A such as combining steps 7 and 8. Appendix C gives the output listing of the MPC compiler for a PIC16CXX device using the optimized encoder algorithm and the decoder algorithm from the next section.

DECOMPRESSION

The input into the decoder, *t*, must be an 8-bit number containing the 4-bit ADPCM data in sign-magnitude format. The range of allowable values for *t* is 0 to 15, where 7 = 0x07 and -7 = 0x0F. Figure 2 shows a block diagram for ADPCM decompression and Appendix B has a listing of the function **`ADPCMDecoder()`**. The predicted sample, s_p, and the quantizer step size index are saved in a structure for the next iteration of the decoder. Initially, the quantizer step size index and the predicted sample (s_p) are set to zero. This function takes a 4-bit sign-magnitude ADPCM code and returns the 16-bit two's complement speech sample.

This decoder is the same as the one used in the encoder routine. It uses the ADPCM value to update the inverse quantizer, which produces a difference, d_q. The difference, d_q, is added to the predicted sample, s_p, to produce the output sample, s_r. The output sample, s_r, is then saved into the predicted sample, s_p, for the next iteration of the decoder.

FIGURE 2: ADPCM DECODER BLOCK DIAGRAM



The following (Table 2) is a step-by-step description of the **ADPCMDecoder()** function from Appendix B.

TABLE 2: ADPCMDecoder() STEP-BY-STEP FUNCTIONS

1. ADPCMDecoder takes an 8-bit number containing the 4-bit ADPCM code (0-15) and returns a 16-bit signed number (speech sample, 32767 to -32768)	`signed long ADPCMDecoder( char code )`
2. Restore the previous values of predicted sample (s_p) and quantizer step size index	`predsample = state.prevsample;` `index = state.previndex;`
3. Find the quantizer step size (q) from a table lookup using the quantizer step size index	`step = StepSizeTable[index];`

4. Inverse quantize the ADPCM code (l) into a predicted difference (d_q) using the quantizer step size (q)	`diffq = step >> 3;`
	`if(code & 4)`
	`  diffq += step;`
	`if(code & 2)`
	`  diffq += step >> 1;`
	`if(code & 1)`
	`  diffq += step >> 2;`

5. Fixed predictor computes new predicted sample (s_r) by adding the old predicted sample (s_p) to the predicted difference (d_q)	`if(code & 8)`
	`  predsample -= diffq;`
	`else`
	`  predsample += diffq;`

6. Check for overflow of the new predicted sample (s_r). s_r, which is a signed 16-bit sample, must be in the range of 32767 to -32768	`if(predsample > 32767)`
	`  predsample = 32767;`
	`else if(predsample < -32768)`
	`  predsample = -32768;`

7. Find the new quantizer step size (q) by adding the previous index and a table lookup using the ADPCM code (l)	`index += IndexTable[code];`

8. Check for overflow of the new quantizer step size index	`if(index < 0)`
	`  index = 0;`
	`if(index > 88)`
	`  index = 88;`

9. Save the new predicted sample (s_r) and quantizer step size index for next iteration	`state.prevsample = predsample;` `state.previndex = index;`
10. Return the new sample (s_r)	`return ( predsample );`

This function requires three 16-bit variables and one 8-bit variable. Appendix C gives the listing output of the MP-C compiler of the decoder algorithm and the optimized encoder algorithm.

© 1997 Microchip Technology Inc.

IMA ADPCM REFERENCE ALGORITHM

The IMA, specifically the Digital Audio Technical Working Group, is a trade association with representatives from companies such as Compaq®, Apple Computer®, Crystal Semiconductor®, DEC®, Hewlett-Packard®, Intel®, Microsoft®, Sony®, and Texas Instruments® to name a few. This group is working towards a standard that defines the exchange of high quality audio data between computing platforms. The algorithm from Intel/DVI® (Digital Video Interactive) has been selected as the standard due to it's audio dynamic range and low data rate. The recommended digital audio exchange formats are given in Table 3.

The algorithms that are implemented in this application note were derived from the IMA ADPCM Reference algorithm. The data format is 8.0 kHz, mono, 4-bit ADPCM. Essentially, the compression and decompression use an adaptive quantization with fixed prediction. The adaptive quantization are based on a table lookup first developed by Intel/DVI for the IMA. Appendix D gives the information about IMA and the supporting documentation for IMA ADPCM.

TABLE 3: DIGITAL AUDIO EXCHANGE FORMATS

Sampling Rate	Mono/Stereo	Data Format	Notes
8.0 kHz	mono	8-bit μ-Law PCM	CCITT G.711 Standard
	mono	8-bit A-Law PCM	CCITT G.711 Standard
	mono	4-bit ADPCM	DVI Algorithm
11.025 kHz	mono/stereo	8-bit Linear PCM	Macintosh® & MP-C Standard
	mono/stereo	4-bit ADPCM	DVI Algorithm
22.05 kHz	mono/stereo	8-bit Linear PCM	Macintosh & MPC Standard
	mono/stereo	4-bit ADPCM	DVI Algorithm
44.10 kHz	mono/stereo	16-bit Linear PCM	CD-DA Standard
	mono/stereo	4-bit ADPCM	DVI Algorithm

PERFORMANCE

Table 4 shows the performance comparison of the ADPCM compression and decompression routines for the PIC16C5X, PIC16CXXX, and PIC17CXXX family of devices assuming 80 additional instruction cycles for overhead. Any device without a PWM module will have to increase the operating frequency to generate a software PWM. Table 4 also provides the minimum external operating frequency required to run the routines. The C code from Appendix C was used. The input/output data rate of the speech samples is 8.0 kHz. The minimum operating frequency is calculated as follows:

8000 x # of Total Instruction Cycles x 4.

For example, the ADPCM encoding using the PIC16CXXX family would require an 8000 x (225 + 80) x 4 = 9.760 MHz external crystal.

TABLE 4: PERFORMANCE COMPARISON TABLE

Device	Encode/Decode	# of Instruction Cycles	Minimum External Operating Frequency
PIC16C5X	Encode	273	11.296 MHz
	Decode	208	9.216 MHz
PIC16CXXX	Encode	225	9.760 MHz
	Decode	168	7.936 MHz
PIC17CXXX	Encode	199	8.928 MHz
	Decode	161	7.712 MHz

Compaq is a registered trademark of Compaq Computer.
Apple Computer and Macintosh are a registered trademarks of Apple Computer, Inc.
Crystal Semiconductor is a registered trademark of Crystal Semiconductor.
DEC is a registered trademark of Digital Equipment Corporation.
Hewlett-Packard is a registered trademark of Hewlett-Packard Company.
Intel and DVI are registered trademarks of Intel Corporation.
Microsoft is a registered trademark of Microsoft Corporation.
Sony is a registered trademark of Sony Corporation.

© 1997 Microchip Technology Inc.

Table 5 illustrates the amount of program and data memory that the ADPCM algorithms consume for the various PICmicro devices. The program memory numbers may change slightly depending on the specific device being used. The table memory column shows how much program memory is used to store the two lookup tables used by the ADPCM algorithm.

TABLE 5: DEVICE MEMORY CONSUMED

Device		Program Memory (words)	Data Memory (bytes)	Table Lookup (words)
PIC16C5X	Encode	273	13	196
	Decode	205	10	
PIC16CXXX	Encode	220	13	196
	Decode	162	10	
PIC17CXXX	Encode	203	13	97
	Decode	164	10	

APPLICATION

The hardware for this application note implements only the decompression algorithm, but the compression algorithm is also included in the firmware. A block diagram is shown in Figure 3 and the schematic is provided in Appendix E. The complete source code listing, including a block diagram, is given in Appendix F.

The board uses a PIC16C72, rated for 20 MHz operation, to control the speech decoding and output. The A/D (Analog-to-Digital) converter of the PIC16C72 is not used in this design. Two 27C512A EPROMs from Microchip Technology are used to store up to 32.768 seconds of ADPCM data. Each EPROM holds 65536 bytes. Each byte in the EPROM contains two ADPCM values. One second of speech requires 8000 ADPCM codes (8 kHz sample rate). Therefore, each EPROM holds (65536 x 2) / 8000 = 16.384 seconds of speech. A 16-bit up counter is used to clock data out of the EPROMs. This method uses only two I/O lines to control the counter and eleven I/O lines to read the EPROM data.

Speech regeneration is accomplished by using the Capture/Compare/PWM (CCP) module of the PIC16C72. The PWM module is configured for a period of 32 kHz. This allows each sample of the speech signal to be output for four PWM periods. The period is calculated by using the following equation from Section 10.3 of the PIC16C7X Data Sheet (DS30390). From the following calculation, PR2 is set to a value of 155.

$$\text{PWM period} = [\text{PR2} + 1] * 4 * \text{Tosc} * (\text{TMR2 prescale value})$$

$$1 / 32\ \text{kHz} = [\text{PR2} + 1] * 4 * (1 / 20\ \text{MHz}) * 1$$
$$31.25\ \mu\text{s} = [\text{PR2} + 1] * 4 * 50\ \text{ns} * 1$$
$$156.25 = \text{PR2} + 1$$
$$155.25 = \text{PR2}$$

The CCP module has the capability of up to 10-bit resolution for the PWM duty cycle. The maximum resolution of the duty cycle that can be achieved for a 32 kHz period can be calculated using the following equation from Section 10.3 of the PIC16C7X Data Sheet.

$$\text{PWM duty cycle} = \text{DC<10:0>} * \text{Tosc} * (\text{TMR2 prescale value})$$

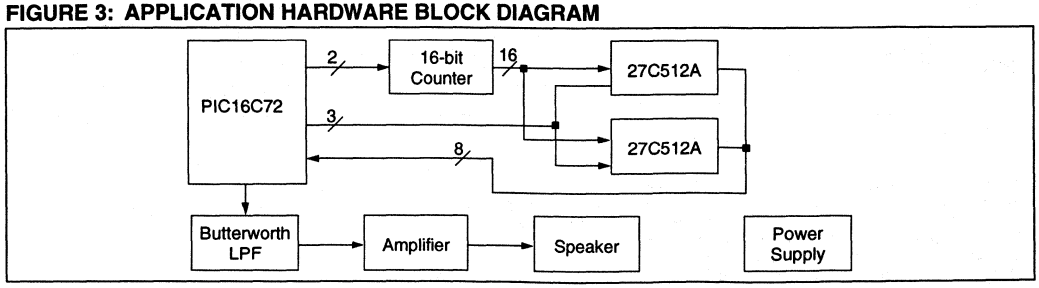
where DC<10:0> = 2^x where x = bits of resolution

$$1 / 32\ \text{kHz} = 2^x * (1 / 20\ \text{MHz}) * 1$$
$$31.25\ \mu\text{s} = 2^x * 50\ \text{ns} * 1$$
$$625 = 2^x$$
$$\log(625) = \log(2^x)$$
$$\log(625) = x * \log(2)$$
$$9.3 = x$$

A PWM duty cycle with up to 9-bits of resolution may be used with a period of 32 kHz. The upper 9-bits of each speech sample are used to set the PWM duty cycle (CCPR1L = sample<15:9>, CCP1CON<5:4> = sample<8:7>). This PWM speech signal is passed through a 4th-order Butterworth low pass filter with a corner frequency of 4 kHz. The low pass filter converts the PWM into an analog voltage level. Finally, the analog voltage level is amplified by a National Semiconductor LM386N-3 before entering the speaker.

Every 125 μs (8 kHz) one ADPCM code must be converted into a speech sample for output to the PWM. This frame of 125 μs relates to 625 instruction cycles for an external crystal frequency of 20 MHz. The ADPCM decode routine, EPROM reads and writes to the PWM must be accomplished within 625 instruction cycles.

FIGURE 3: APPLICATION HARDWARE BLOCK DIAGRAM



© 1997 Microchip Technology Inc.

To implement audio recording, the following changes to the application hardware would have to be made:

- Replace the EPROMs with RAM and
- Add a microphone with input filtering and
- Use the on-board A/D converter of the PIC16C7X devices for 8-bit recording or
- Use an external 12- to 16-bit A/D converter for better quality recording

The following is the sequence of events to record and store speech data. A timer would be used to set the sample rate. When the timer overflowed an A/D conversion would be started.

1. Start an A/D conversion when the timer overflows.
2. Read the sample and call the routine `ADPCMEncoder()`.
3. Store the result in the upper 4-bits of a temporary variable.
4. Start an A/D conversion when the timer overflows.
5. Read the sample and call the routine `ADPCMEncoder()`.
6. Store the result in the lower 4-bits of the temporary variable.
7. Write the temporary variable to the SRAM.
8. GOTO 1.

Typical conversion times for the PIC16C7XX devices are 40 μs. Assuming that the external crystal frequency is 20 MHz and the sample rate is 8 kHz, the encoder routine takes approximately 45 μs (225 instruction cycles x 200 ns) to complete. This leaves approximately 40 μs or 200 instruction cycles to read the A/D converter, write to the SRAM and complete any other tasks. An external A/D converter that is faster could be used to increase the amount of time for processing other tasks.

COMPUTER PROGRAM

The IMA Reference ADPCM algorithm was written for a PC using Borland C++ version 3.1. This program uses the same routines that are used in Appendix C. A raw sound file is recorded on the PC using a 16-bit sound card. Speech is recorded at 8.0 kHz in 16-bit mono mode. This raw speech file is processed by the encoder part of the program which creates a file with the ADPCM values. This file can now be burned into EPROMs for use with the application hardware. The decoder part of the program can take this file and create a new 16-bit raw speech file. The 16-bit sound card on the PC can play this file to ensure that the ADPCM compression/decompression routines are working properly. The PC program is given in Appendix F. It is composed of three files: `pcspeech.c`, `pcadpcm.c`, and `pcadpcm.h`.

CONCLUSION

The final results of the application hardware using the PIC16C72 are:

- Compression and Decompression
  - 906 words of Program Memory used
  - 44% of total Program Memory
  - 24 bytes of Data Memory used
  - 19% of total Data Memory
- Decompression only
  - 686 words of Program Memory used
  - 33% of total Program Memory
  - 22 bytes of Data Memory used
  - 17% of total Data Memory
- ~250 Instruction cycles to decode one ADPCM code
  - 40% of frame (250/625)
- Hardware used
  - CCP1 configured for PWM (9-bit duty cycle, 32 kHz period speech output)
  - Timer2 to set the 8.0 kHz output sample rate
  - 13 I/O lines to read data from the EPROMs

References:

1. *Recommended Practices for Enhancing Digital Audio Compatibility in Multimedia Systems*, Revision 3.00, Interactive Multimedia Association, October 21, 1992.
2. *Digital Audio Special Edition Proceedings*, Volume 2, Issue 2, Interactive Multimedia Association, May 1992.
3. Panos E. Papamichalis Ph.D., *Practical Approaches to Speech Coding*, Prentice-Hall Inc., Englewood Cliffs, N.J, 1987.
4. Adaptive Differential Pulse-Code Modulation, *Digital Signal Processing Applications using the ADSP-2100 Family*, Volume 1, Analog Devices, Prentice-Hall, Englewood Cliffs, N.J., 1992.
5. 32-kbits/s ADPCM with the TMS32010, *Digital Signal Processing Applications with the TMS320 Family*, SPRA012, Jay Reimer, Mike McMahan, and Masud Arjmand, Texas Instruments, 1986.

© 1997 Microchip Technology Inc.

APPENDIX A: ADPCMEncoder() FUNCTION

```
/* Table of index changes */
const int IndexTable[16] = {
    0xff, 0xff, 0xff, 0xff, 2, 4, 6, 8,
    0xff, 0xff, 0xff, 0xff, 2, 4, 6, 8
};

/* Quantizer step size lookup table */
const long StepSizeTable[89] = {
    7, 8, 9, 10, 11, 12, 13, 14, 16, 17,
    19, 21, 23, 25, 28, 31, 34, 37, 41, 45,
    50, 55, 60, 66, 73, 80, 88, 97, 107, 118,
    130, 143, 157, 173, 190, 209, 230, 253, 279, 307,
    337, 371, 408, 449, 494, 544, 598, 658, 724, 796,
    876, 963, 1060, 1166, 1282, 1411, 1552, 1707, 1878, 2066,
    2272, 2499, 2749, 3024, 3327, 3660, 4026, 4428, 4871, 5358,
    5894, 6484, 7132, 7845, 8630, 9493, 10442, 11487, 12635, 13899,
    15289, 16818, 18500, 20350, 22385, 24623, 27086, 29794, 32767
};

signed long diff;               /* Difference between sample and predicted sample */
long step;                      /* Quantizer step size */
signed long predsample;         /* Output of ADPCM predictor */
signed long diffq;              /* Dequantized predicted difference */
int index;                      /* Index into step size table */

/*****************************************************************************
*       ADPCMEncoder - ADPCM encoder routine                                 *
*****************************************************************************
*       Input Variables:                                                     *
*               signed long sample - 16-bit signed speech sample             *
*       Return Variable:                                                     *
*               char - 8-bit number containing the 4-bit ADPCM code          *
*****************************************************************************/
char ADPCMEncoder( signed long sample )
{
    int code;               /* ADPCM output value */
    int tempstep;           /* Temporary step size */

    /* Restore previous values of predicted sample and quantizer step
       size index
    */
    predsample = state.prevsample;
    index = state.previndex;
    step = StepSizeTable[index];


    /* Compute the difference between the actual sample (sample) and the
       the predicted sample (predsample)
    */
    diff = sample - predsample;
    if(diff >= 0)
        code = 0;
    else
    {
        code = 8;
        diff = -diff;
    }

    /* Quantize the difference into the 4-bit ADPCM code using the
       the quantizer step size
    */
    tempstep = step;
    if( diff >= tempstep )
    {
        code |= 4;
```


© 1997 Microchip Technology Inc.

```
        diff -= tempstep;
    }
    tempstep >>= 1;
    if( diff >= tempstep )
    {
        code |= 2;
        diff -= tempstep;
    }
    tempstep >>= 1;
    if( diff >= tempstep )
        code |= 1;

    /* Inverse quantize the ADPCM code into a predicted difference
       using the quantizer step size
    */
    diffq = step >> 3;
    if( code & 4 )
        diffq += step;
    if( code & 2 )
        diffq += step >> 1;
    if( code & 1 )
        diffq += step >> 2;

    /* Fixed predictor computes new predicted sample by adding the
       old predicted sample to predicted difference
    */
    if( code & 8 )
        predsample -= diffq;
    else
        predsample += diffq;

    /* Check for overflow of the new predicted sample
    */
    if( predsample > 32767 )
        predsample = 32767;
    else if( predsample < -32768 )
        predsample = -32768;

    /* Find new quantizer stepsize index by adding the old index
       to a table lookup using the ADPCM code
    */
    index += IndexTable[code];

    /* Check for overflow of the new quantizer step size index
    */
    if( index < 0 )
        index = 0;
    if( index > 88 )
        index = 88;

    /* Save the predicted sample and quantizer step size index for
       next iteration
    */
    state.prevsample = predsample;
    state.previndex = index;

    /* Return the new ADPCM code */
    return ( code & 0x0f );
}
```


© 1997 Microchip Technology Inc.

APPENDIX B: ADPCMDecoder() FUNCTION

```
/* Table of index changes */
const int IndexTable[16] = {
    0xff, 0xff, 0xff, 0xff, 2, 4, 6, 8,
    0xff, 0xff, 0xff, 0xff, 2, 4, 6, 8
};

/* Quantizer step size lookup table */
const long StepSizeTable[89] = {
    7, 8, 9, 10, 11, 12, 13, 14, 16, 17,
    19, 21, 23, 25, 28, 31, 34, 37, 41, 45,
    50, 55, 60, 66, 73, 80, 88, 97, 107, 118,
    130, 143, 157, 173, 190, 209, 230, 253, 279, 307,
    337, 371, 408, 449, 494, 544, 598, 658, 724, 796,
    876, 963, 1060, 1166, 1282, 1411, 1552, 1707, 1878, 2066,
    2272, 2499, 2749, 3024, 3327, 3660, 4026, 4428, 4871, 5358,
    5894, 6484, 7132, 7845, 8630, 9493, 10442, 11487, 12635, 13899,
    15289, 16818, 18500, 20350, 22385, 24623, 27086, 29794, 32767
};

long step;                      /* Quantizer step size */
signed long predsample;         /* Output of ADPCM predictor */
signed long diffq;              /* Dequantized predicted difference */
int index;                      /* Index into step size table */

/*******************************************************************************
*       ADPCMDecoder - ADPCM decoder routine                                   *
*******************************************************************************
*       Input Variables:                                                       *
*               char code - 8-bit number containing the 4-bit ADPCM code       *
*       Return Variable:                                                       *
*               signed long - 16-bit signed speech sample                      *
*******************************************************************************/
signed long ADPCMDecoder(char code )
{

    /* Restore previous values of predicted sample and quantizer step
       size index
    */
    predsample = state.prevsample;
    index = state.previndex;

    /* Find quantizer step size from lookup table using index
    */
    step = StepSizeTable[index];

    /* Inverse quantize the ADPCM code into a difference using the
       quantizer step size
    */
    diffq = step >> 3;
    if( code & 4 )
        diffq += step;
    if( code & 2 )
        diffq += step >> 1;
    if( code & 1 )
        diffq += step >> 2;

    /* Add the difference to the predicted sample
    */
    if( code & 8 )
        predsample -= diffq;
    else
        predsample += diffq;

    /* Check for overflow of the new predicted sample
```


© 1997 Microchip Technology Inc.

```
    */
    if( predsample > 32767 )
        predsample = 32767;
    else if( predsample < -32768 )
        predsample = -32768;

    /* Find new quantizer step size by adding the old index and a
       table lookup using the ADPCM code
    */
    index += IndexTable[code];

    /* Check for overflow of the new quantizer step size index
    */
    if( index < 0 )
        index = 0;
    if( index > 88 )
        index = 88;


    /* Save predicted sample and quantizer step size index for next
       iteration
    */
    state.prevsample = predsample;
    state.previndex = index;

    /* Return the new speech sample */
    return( predsample );

}
```


© 1997 Microchip Technology Inc.

APPENDIX C: OPTIMIZED `ADPCMEncoder()`, `ADPCMDecoder()`, AND TABLE LISTING

```
/************************************************************************
*    Filename:    ADPCM.C                                                *
************************************************************************
*    Author:      Rodger Richey                                          *
*    Company:     Microchip Technology Incorporated                      *
*    Revision:    0                                                      *
*    Date:        1-9-96                                                 *
*    Compiled using Bytecraft Ltd. MPC version BC.193                    *
************************************************************************
*    This file contains the ADPCM encode and decode routines. This      *
*    routines were obtained from the Interactive Multimedia Association's *
*    Reference ADPCM algorithm. This algorithm was first implemented by  *
*    Intel/DVI.                                                          *
************************************************************************/
                                /* Table of index changes */
                                const int IndexTable[16] = {0xff,0xff,0xff,0xff,2,4,6,8,
                                    0xff, 0xff, 0xff, 0xff, 2, 4, 6, 8
0005 0782    ADDWF  PCL         };
0006 34FF    RETLW  FFh
0007 34FF    RETLW  FFh
0008 34FF    RETLW  FFh
0009 34FF    RETLW  FFh
000A 3402    RETLW  02h
000B 3404    RETLW  04h
000C 3406    RETLW  06h
000D 3408    RETLW  08h
000E 34FF    RETLW  FFh
000F 34FF    RETLW  FFh
0010 34FF    RETLW  FFh
0011 34FF    RETLW  FFh
0012 3402    RETLW  02h
0013 3404    RETLW  04h
0014 3406    RETLW  06h
0015 3408    RETLW  08h
                                /* Quantizer step size lookup table */
                                const long StepSizeTable[89] = {
                                    7,8,9,10,11,12,13,14,16,17,19,21,23,25,28,31,34,37,
                                    41,45,50,55,60,66,73,80,88,97,107,118,130,143,157,
                                    173,190,209,230,253,279,307,337,371,408,449,494,544,
                                    598,658,724,796,876,963,1060,1166,1282,1411,1552,
                                    1707,1878,2066,2272,2499,2749,3024,3327,3660,4026,
                                    4428,4871,5358,5894,6484,7132,7845,8630,9493,10442,
                                    11487,12635,13899,15289,16818,18500,20350,22385,
                                    24623, 27086, 29794, 32767
0016 0782    ADDWF  PCL         };
0017 3407    RETLW  07h
0018 3400    RETLW  00h
0019 3408    RETLW  08h
001A 3400    RETLW  00h
001B 3409    RETLW  09h
001C 3400    RETLW  00h
001D 340A    RETLW  0Ah
001E 3400    RETLW  00h
001F 340B    RETLW  0Bh
0020 3400    RETLW  00h
0021 340C    RETLW  0Ch
0022 3400    RETLW  00h
0023 340D    RETLW  0Dh
0024 3400    RETLW  00h
0025 340E    RETLW  0Eh
0026 3400    RETLW  00h
0027 3410    RETLW  10h
0028 3400    RETLW  00h
0029 3411    RETLW  11h
002A 3400    RETLW  00h
```


© 1997 Microchip Technology Inc.

```
002B 3413    RETLW  13h
002C 3400    RETLW  00h
002D 3415    RETLW  15h
002E 3400    RETLW  00h
002F 3417    RETLW  17h
0030 3400    RETLW  00h
0031 3419    RETLW  19h
0032 3400    RETLW  00h
0033 341C    RETLW  1Ch
0034 3400    RETLW  00h
0035 341F    RETLW  1Fh
0036 3400    RETLW  00h
0037 3422    RETLW  22h
0038 3400    RETLW  00h
0039 3425    RETLW  25h
003A 3400    RETLW  00h
003B 3429    RETLW  29h
003C 3400    RETLW  00h
003D 342D    RETLW  2Dh
003E 3400    RETLW  00h
003F 3432    RETLW  32h
0040 3400    RETLW  00h
0041 3437    RETLW  37h
0042 3400    RETLW  00h
0043 343C    RETLW  3Ch
0044 3400    RETLW  00h
0045 3442    RETLW  42h
0046 3400    RETLW  00h
0047 3449    RETLW  49h
0048 3400    RETLW  00h
0049 3450    RETLW  50h
004A 3400    RETLW  00h
004B 3458    RETLW  58h
004C 3400    RETLW  00h
004D 3461    RETLW  61h
004E 3400    RETLW  00h
004F 346B    RETLW  6Bh
0050 3400    RETLW  00h
0051 3476    RETLW  76h
0052 3400    RETLW  00h
0053 3482    RETLW  82h
0054 3400    RETLW  00h
0055 348F    RETLW  8Fh
0056 3400    RETLW  00h
0057 349D    RETLW  9Dh
0058 3400    RETLW  00h
0059 34AD    RETLW  ADh
005A 3400    RETLW  00h
005B 34BE    RETLW  BEh
005C 3400    RETLW  00h
005D 34D1    RETLW  D1h
005E 3400    RETLW  00h
005F 34E6    RETLW  E6h
0060 3400    RETLW  00h
0061 34FD    RETLW  FDh
0062 3400    RETLW  00h
0063 3417    RETLW  17h
0064 3401    RETLW  01h
0065 3433    RETLW  33h
0066 3401    RETLW  01h
0067 3451    RETLW  51h
0068 3401    RETLW  01h
0069 3473    RETLW  73h
006A 3401    RETLW  01h
006B 3498    RETLW  98h
006C 3401    RETLW  01h
006D 34C1    RETLW  C1h
006E 3401    RETLW  01h
006F 34EE    RETLW  EEh
```


© 1997 Microchip Technology Inc.

```
0070 3401    RETLW  01h
0071 3420    RETLW  20h
0072 3402    RETLW  02h
0073 3456    RETLW  56h
0074 3402    RETLW  02h
0075 3492    RETLW  92h
0076 3402    RETLW  02h
0077 34D4    RETLW  D4h
0078 3402    RETLW  02h
0079 341C    RETLW  1Ch
007A 3403    RETLW  03h
007B 346C    RETLW  6Ch
007C 3403    RETLW  03h
007D 34C3    RETLW  C3h
007E 3403    RETLW  03h
007F 3424    RETLW  24h
0080 3404    RETLW  04h
0081 348E    RETLW  8Eh
0082 3404    RETLW  04h
0083 3402    RETLW  02h
0084 3405    RETLW  05h
0085 3483    RETLW  83h
0086 3405    RETLW  05h
0087 3410    RETLW  10h
0088 3406    RETLW  06h
0089 34AB    RETLW  ABh
008A 3406    RETLW  06h
008B 3456    RETLW  56h
008C 3407    RETLW  07h
008D 3412    RETLW  12h
008E 3408    RETLW  08h
008F 34E0    RETLW  E0h
0090 3408    RETLW  08h
0091 34C3    RETLW  C3h
0092 3409    RETLW  09h
0093 34BD    RETLW  BDh
0094 340A    RETLW  0Ah
0095 34D0    RETLW  D0h
0096 340B    RETLW  0Bh
0097 34FF    RETLW  FFh
0098 340C    RETLW  0Ch
0099 344C    RETLW  4Ch
009A 340E    RETLW  0Eh
009B 34BA    RETLW  BAh
009C 340F    RETLW  0Fh
009D 344C    RETLW  4Ch
009E 3411    RETLW  11h
009F 3407    RETLW  07h
00A0 3413    RETLW  13h
00A1 34EE    RETLW  EEh
00A2 3414    RETLW  14h
00A3 3406    RETLW  06h
00A4 3417    RETLW  17h
00A5 3454    RETLW  54h
00A6 3419    RETLW  19h
00A7 34DC    RETLW  DCh
00A8 341B    RETLW  1Bh
00A9 34A5    RETLW  A5h
00AA 341E    RETLW  1Eh
00AB 34B6    RETLW  B6h
00AC 3421    RETLW  21h
00AD 3415    RETLW  15h
00AE 3425    RETLW  25h
00AF 34CA    RETLW  CAh
00B0 3428    RETLW  28h
00B1 34DF    RETLW  DFh
00B2 342C    RETLW  2Ch
00B3 345B    RETLW  5Bh
00B4 3431    RETLW  31h
```


© 1997 Microchip Technology Inc.

```
00B5 344B    RETLW  4Bh
00B6 3436    RETLW  36h
00B7 34B9    RETLW  B9h
00B8 343B    RETLW  3Bh
00B9 34B2    RETLW  B2h
00BA 3441    RETLW  41h
00BB 3444    RETLW  44h
00BC 3448    RETLW  48h
00BD 347E    RETLW  7Eh
00BE 344F    RETLW  4Fh
00BF 3471    RETLW  71h
00C0 3457    RETLW  57h
00C1 342F    RETLW  2Fh
00C2 3460    RETLW  60h
00C3 34CE    RETLW  CEh
00C4 3469    RETLW  69h
00C5 3462    RETLW  62h
00C6 3474    RETLW  74h
00C7 34FF    RETLW  FFh
00C8 347F    RETLW  7Fh
```

```
002E                            signed long diff; /* Difference between sample and
                                                     predicted sample */
0030                            long step;                    /* Quantizer step size */
0032                            signed long predsample;  /* Output of ADPCM predictor */
0034                            signed long diffq;/* Dequantized predicted difference */
0036                            int index;              /* Index into step size table */

/*****************************************************************************
*    ADPCMEncoder - ADPCM encoder routine                                    *
*****************************************************************************
*    Input Variables:                                                        *
*       signed long sample - 16-bit signed speech sample                     *
*    Return Variable:                                                        *
*       char - 8-bit number containing the 4-bit ADPCM code                  *
*****************************************************************************/
                                char ADPCMEncoder( signed long sample )
0037                            {
00C9 1283    BCF    STATUS,RP0
00CA 00B7    MOVWF  37
00CB 0804    MOVF   FSR,W
00CC 00B8    MOVWF  38
0039                                int code;               /* ADPCM output value */

                                    /* Restore previous values of predicted sample and
                                       quantizer step size index */
00CD 082B    MOVF   2B,W            predsample = state.prevsample;
00CE 00B2    MOVWF  32
00CF 082C    MOVF   2C,W
00D0 00B3    MOVWF  33
00D1 082D    MOVF   2D,W            index = state.previndex;
00D2 00B6    MOVWF  36
00D3 0836    MOVF   36,W            step = StepSizeTable[index];
00D4 0084    MOVWF  FSR
00D5 1003    BCF    STATUS,C
00D6 0D84    RLF    FSR
00D7 3000    MOVLW  00h
00D8 008A    MOVWF  PCLATH
00D9 0804    MOVF   FSR,W
00DA 2016    CALL   0016h
00DB 1283    BCF    STATUS,RP0
00DC 00B0    MOVWF  30
00DD 0A84    INCF   FSR
00DE 3000    MOVLW  00h
00DF 008A    MOVWF  PCLATH
00E0 0804    MOVF   FSR,W
00E1 2016    CALL   0016h
00E2 1283    BCF    STATUS,RP0
00E3 00B1    MOVWF  31
```

```
                                        /* Compute the difference between the actual sample
                                          (sample) and the predicted sample (predsample) */
00E4 0832    MOVF   32,W                diff = sample - predsample;
00E5 0237    SUBWF  37,W
00E6 00AE    MOVWF  2E
00E7 0833    MOVF   33,W
00E8 1C03    BTFSS  STATUS,C
00E9 3E01    ADDLW  01h
00EA 0238    SUBWF  38,W
00EB 00AF    MOVWF  2F
00EC 1BAF    BTFSC  2F,7                if(diff >= 0)
00ED 28F0    GOTO   00F0h
00EE 01B9    CLRF   39                     code = 0;
00EF 28F9    GOTO   00F9h               else
                                        {
00F0 3008    MOVLW  08h                    code = 8;
00F1 00B9    MOVWF  39
00F2 09AF    COMF   2F                     diff = -diff;
00F3 09AE    COMF   2E
00F4 0AAE    INCF   2E
00F5 1D03    BTFSS  STATUS,Z
00F6 28F9    GOTO   00F9h
00F7 1283    BCF    STATUS,RP0
00F8 0AAF    INCF   2F
                                        }
                                        /* Quantize the difference into the 4-bit ADPCM code
                                           using the quantizer step size. Inverse quantize
                                           the ADPCM code into a predicted difference using
                                           the quantizer step size */
00F9 1283    BCF    STATUS,RP0          diffq = step >> 3;
00FA 0830    MOVF   30,W
00FB 00B4    MOVWF  34
00FC 0831    MOVF   31,W
00FD 00B5    MOVWF  35
00FE 0D35    RLF    35,W
00FF 0CB5    RRF    35
0100 0CB4    RRF    34
0101 0D35    RLF    35,W
0102 0CB5    RRF    35
0103 0CB4    RRF    34
0104 0D35    RLF    35,W
0105 0CB5    RRF    35
0106 0CB4    RRF    34
0107 0830    MOVF   30,W                if( diff >= step )
0108 022E    SUBWF  2E,W
0109 3080    MOVLW  80h
010A 062F    XORWF  2F,W
010B 00A9    MOVWF  29
010C 3080    MOVLW  80h
010D 0631    XORWF  31,W
010E 1C03    BTFSS  STATUS,C
010F 3E01    ADDLW  01h
0110 0229    SUBWF  29,W
0111 1C03    BTFSS  STATUS,C
0112 2921    GOTO   0121h
0113                                    {
0113 1283    BCF    STATUS,RP0             code |= 4;
0114 1539    BSF    39,2
0115 0830    MOVF   30,W                   diff -= step;
0116 02AE    SUBWF  2E
0117 0831    MOVF   31,W
0118 1C03    BTFSS  STATUS,C
0119 3E01    ADDLW  01h
011A 02AF    SUBWF  2F
011B 0830    MOVF   30,W                   diffq += step;
011C 07B4    ADDWF  34
011D 0831    MOVF   31,W
011E 1803    BTFSC  STATUS,C
011F 3E01    ADDLW  01h
```


© 1997 Microchip Technology Inc.

```
0120 07B5    ADDWF  35
                                    }
0121 1283    BCF    STATUS,RP0      step >>= 1;
0122 0D31    RLF    31,W
0123 0CB1    RRF    31
0124 0CB0    RRF    30
0125 0830    MOVF   30,W            if( diff >= step )
0126 022E    SUBWF  2E,W
0127 3080    MOVLW  80h
0128 062F    XORWF  2F,W
0129 00A9    MOVWF  29
012A 3080    MOVLW  80h
012B 0631    XORWF  31,W
012C 1C03    BTFSS  STATUS,C
012D 3E01    ADDLW  01h
012E 0229    SUBWF  29,W
012F 1C03    BTFSS  STATUS,C
0130 293F    GOTO   013Fh
0131                                {
0131 1283    BCF    STATUS,RP0         code |= 2;
0132 14B9    BSF    39,1
0133 0830    MOVF   30,W               diff -= step;
0134 02AE    SUBWF  2E
0135 0831    MOVF   31,W
0136 1C03    BTFSS  STATUS,C
0137 3E01    ADDLW  01h
0138 02AF    SUBWF  2F
0139 0830    MOVF   30,W               diffq += step;
013A 07B4    ADDWF  34
013B 0831    MOVF   31,W
013C 1803    BTFSC  STATUS,C
013D 3E01    ADDLW  01h
013E 07B5    ADDWF  35
                                    }
013F 1283    BCF    STATUS,RP0      step >>= 1;
0140 0D31    RLF    31,W
0141 0CB1    RRF    31
0142 0CB0    RRF    30
0143 0830    MOVF   30,W            if( diff >= step )
0144 022E    SUBWF  2E,W
0145 3080    MOVLW  80h
0146 062F    XORWF  2F,W
0147 00A9    MOVWF  29
0148 3080    MOVLW  80h
0149 0631    XORWF  31,W
014A 1C03    BTFSS  STATUS,C
014B 3E01    ADDLW  01h
014C 0229    SUBWF  29,W
014D 1C03    BTFSS  STATUS,C
014E 2957    GOTO   0157h
014F                                {
014F 1283    BCF    STATUS,RP0         code |= 1;
0150 1439    BSF    39,0
0151 0830    MOVF   30,W               diffq += step;
0152 07B4    ADDWF  34
0153 0831    MOVF   31,W
0154 1803    BTFSC  STATUS,C
0155 3E01    ADDLW  01h
0156 07B5    ADDWF  35
                                    }
                                    /* Fixed predictor computes new predicted sample by
                                       adding the old predicted sample to predicted
                                       difference */
0157 1283    BCF    STATUS,RP0      if( code & 8 )
0158 1DB9    BTFSS  39,3
0159 2961    GOTO   0161h
015A 0834    MOVF   34,W               predsample -= diffq;
015B 02B2    SUBWF  32
015C 0835    MOVF   35,W
```

```
015D 1C03    BTFSS  STATUS,C
015E 3E01    ADDLW  01h
015F 02B3    SUBWF  33
0160 2967    GOTO   0167h            else
0161 0834    MOVF   34,W               predsample += diffq;
0162 07B2    ADDWF  32
0163 0835    MOVF   35,W
0164 1803    BTFSC  STATUS,C
0165 3E01    ADDLW  01h
0166 07B3    ADDWF  33
                                     /* Check for overflow of the new predicted sample */
0167 3001    MOVLW  01h              if( predsample > 32767 )
0168 0732    ADDWF  32,W
0169 00A8    MOVWF  28
016A 3080    MOVLW  80h
016B 0633    XORWF  33,W
016C 00A7    MOVWF  27
016D 3000    MOVLW  00h
016E 1803    BTFSC  STATUS,C
016F 3001    MOVLW  01h
0170 0727    ADDWF  27,W
0171 0428    IORWF  28,W
0172 1D03    BTFSS  STATUS,Z
0173 1C03    BTFSS  STATUS,C
0174 297B    GOTO   017Bh
0175 30FF    MOVLW  FFh                predsample = 32767;
0176 1283    BCF    STATUS,RP0
0177 00B2    MOVWF  32
0178 307F    MOVLW  7Fh
0179 00B3    MOVWF  33
017A 2987    GOTO   0187h            else if( predsample < -32768 )
017B 3080    MOVLW  80h                predsample = -32768;
017C 1283    BCF    STATUS,RP0
017D 0633    XORWF  33,W
017E 00A7    MOVWF  27
017F 3000    MOVLW  00h
0180 0227    SUBWF  27,W
0181 1803    BTFSC  STATUS,C
0182 2987    GOTO   0187h
0183 1283    BCF    STATUS,RP0
0184 01B2    CLRF   32
0185 3080    MOVLW  80h
0186 00B3    MOVWF  33
                                     /* Find new quantizer stepsize index by adding the
                                      old index to a table lookup using the ADPCM code */
0187 018A    CLRF   PCLATH           index += IndexTable[code];
0188 1283    BCF    STATUS,RP0
0189 0839    MOVF   39,W
018A 2005    CALL   0005h
018B 1283    BCF    STATUS,RP0
018C 07B6    ADDWF  36
                                     /* Check for overflow of the new quantizer step size
                                        index */
018D 1BB6    BTFSC  36,7             if( index < 0 )
018E 01B6    CLRF   36                 index = 0;
018F 3080    MOVLW  80h              if( index > 88 )
0190 1283    BCF    STATUS,RP0
0191 0636    XORWF  36,W
0192 00A7    MOVWF  27
0193 30D8    MOVLW  D8h
0194 0227    SUBWF  27,W
0195 1D03    BTFSS  STATUS,Z
0196 1C03    BTFSS  STATUS,C
0197 299B    GOTO   019Bh
0198 3058    MOVLW  58h                index = 88;
0199 1283    BCF    STATUS,RP0
019A 00B6    MOVWF  36
                                     /* Save the predicted sample and quantizer step size
                                        index for next iteration */
```


© 1997 Microchip Technology Inc.

```
019B 1283    BCF    STATUS,RP0          state.prevsample = predsample;
019C 0832    MOVF   32,W
019D 00AB    MOVWF  2B
019E 0833    MOVF   33,W
019F 00AC    MOVWF  2C
01A0 0836    MOVF   36,W                state.previndex = index;
01A1 00AD    MOVWF  2D
                                        /* Return the new ADPCM code */
01A2 300F    MOVLW  0Fh                 return ( code & 0x0f );
01A3 0539    ANDWF  39,W
01A4 0008    RETURN
                                    }

/*************************************************************************
*    ADPCMDecoder - ADPCM decoder routine                                *
*************************************************************************
*    Input Variables:                                                    *
*       char code - 8-bit number containing the 4-bit ADPCM code         *
*    Return Variable:                                                    *
*       signed long - 16-bit signed speech sample                        *
*************************************************************************/
                                    signed long ADPCMDecoder(char code )
003A                                {
01A5 1283    BCF    STATUS,RP0
01A6 00BA    MOVWF  3A
                                        /* Restore previous values of predicted sample and
                                           quantizer step size index */
01A7 082B    MOVF   2B,W                predsample = state.prevsample;
01A8 00B2    MOVWF  32
01A9 082C    MOVF   2C,W
01AA 00B3    MOVWF  33
01AB 082D    MOVF   2D,W                index = state.previndex;
01AC 00B6    MOVWF  36
                                        /* Find quantizer step size from lookup table using
                                           index */
01AD 0836    MOVF   36,W                step = StepSizeTable[index];
01AE 0084    MOVWF  FSR
01AF 1003    BCF    STATUS,C
01B0 0D84    RLF    FSR
01B1 3000    MOVLW  00h
01B2 008A    MOVWF  PCLATH
01B3 0804    MOVF   FSR,W
01B4 2016    CALL   0016h
01B5 1283    BCF    STATUS,RP0
01B6 00B0    MOVWF  30
01B7 0A84    INCF   FSR
01B8 3000    MOVLW  00h
01B9 008A    MOVWF  PCLATH
01BA 0804    MOVF   FSR,W
01BB 2016    CALL   0016h
01BC 1283    BCF    STATUS,RP0
01BD 00B1    MOVWF  31
                                        /* Inverse quantize the ADPCM code into a difference
                                           using the quantizer step size */
01BE 0830    MOVF   30,W                diffq = step >> 3;
01BF 00B4    MOVWF  34
01C0 0831    MOVF   31,W
01C1 00B5    MOVWF  35
01C2 0D35    RLF    35,W
01C3 0CB5    RRF    35
01C4 0CB4    RRF    34
01C5 0D35    RLF    35,W
01C6 0CB5    RRF    35
01C7 0CB4    RRF    34
01C8 0D35    RLF    35,W
01C9 0CB5    RRF    35
01CA 0CB4    RRF    34
01CB 1D3A    BTFSS  3A,2                if( code & 4 )
01CC 29D3    GOTO   01D3h
```

```
01CD 0830    MOVF   30,W                 diffq += step;
01CE 07B4    ADDWF  34
01CF 0831    MOVF   31,W
01D0 1803    BTFSC  STATUS,C
01D1 3E01    ADDLW  01h
01D2 07B5    ADDWF  35
01D3 1283    BCF    STATUS,RP0          if( code & 2 )
01D4 1CBA    BTFSS  3A,1
01D5 29E3    GOTO   01E3h
01D6 0830    MOVF   30,W                 diffq += step >> 1;
01D7 00A8    MOVWF  28
01D8 0831    MOVF   31,W
01D9 00A9    MOVWF  29
01DA 0D29    RLF    29,W
01DB 0CA9    RRF    29
01DC 0CA8    RRF    28
01DD 0828    MOVF   28,W
01DE 07B4    ADDWF  34
01DF 0829    MOVF   29,W
01E0 1803    BTFSC  STATUS,C
01E1 3E01    ADDLW  01h
01E2 07B5    ADDWF  35
01E3 1283    BCF    STATUS,RP0          if( code & 1 )
01E4 1C3A    BTFSS  3A,0
01E5 29F6    GOTO   01F6h
01E6 0830    MOVF   30,W                 diffq += step >> 2;
01E7 00A8    MOVWF  28
01E8 0831    MOVF   31,W
01E9 00A9    MOVWF  29
01EA 0D29    RLF    29,W
01EB 0CA9    RRF    29
01EC 0CA8    RRF    28
01ED 0D29    RLF    29,W
01EE 0CA9    RRF    29
01EF 0CA8    RRF    28
01F0 0828    MOVF   28,W
01F1 07B4    ADDWF  34
01F2 0829    MOVF   29,W
01F3 1803    BTFSC  STATUS,C
01F4 3E01    ADDLW  01h
01F5 07B5    ADDWF  35
                                         /* Add the difference to the predicted sample */
01F6 1283    BCF    STATUS,RP0          if( code & 8 )
01F7 1DBA    BTFSS  3A,3
01F8 2A00    GOTO   0200h
01F9 0834    MOVF   34,W                 predsample -= diffq;
01FA 02B2    SUBWF  32
01FB 0835    MOVF   35,W
01FC 1C03    BTFSS  STATUS,C
01FD 3E01    ADDLW  01h
01FE 02B3    SUBWF  33
01FF 2A06    GOTO   0206h                else
0200 0834    MOVF   34,W                 predsample += diffq;
0201 07B2    ADDWF  32
0202 0835    MOVF   35,W
0203 1803    BTFSC  STATUS,C
0204 3E01    ADDLW  01h
0205 07B3    ADDWF  33
                                         /* Check for overflow of the new predicted sample */
0206 3001    MOVLW  01h                  if( predsample > 32767 )
0207 0732    ADDWF  32,W
0208 00A8    MOVWF  28
0209 3080    MOVLW  80h
020A 0633    XORWF  33,W
020B 00A7    MOVWF  27
020C 3000    MOVLW  00h
020D 1803    BTFSC  STATUS,C
020E 3001    MOVLW  01h
020F 0727    ADDWF  27,W
```


© 1997 Microchip Technology Inc.

```
0210 0428    IORWF  28,W
0211 1D03    BTFSS  STATUS,Z
0212 1C03    BTFSS  STATUS,C
0213 2A1A    GOTO   021Ah
0214 30FF    MOVLW  FFh                    predsample = 32767;
0215 1283    BCF    STATUS,RP0
0216 00B2    MOVWF  32
0217 307F    MOVLW  7Fh
0218 00B3    MOVWF  33
0219 2A26    GOTO   0226h                 else if( predsample < -32768 )
021A 3080    MOVLW  80h                    predsample = -32768;
021B 1283    BCF    STATUS,RP0
021C 0633    XORWF  33,W
021D 00A7    MOVWF  27
021E 3000    MOVLW  00h
021F 0227    SUBWF  27,W
0220 1803    BTFSC  STATUS,C
0221 2A26    GOTO   0226h
0222 1283    BCF    STATUS,RP0
0223 01B2    CLRF   32
0224 3080    MOVLW  80h
0225 00B3    MOVWF  33

                                          /* Find new quantizer step size by adding the old
                                             index and a table lookup using the ADPCM code */
0226 018A    CLRF   PCLATH                index += IndexTable[code];
0227 1283    BCF    STATUS,RP0
0228 083A    MOVF   3A,W
0229 2005    CALL   0005h
022A 1283    BCF    STATUS,RP0
022B 07B6    ADDWF  36

                                          /* Check for overflow of the new quantizer step size
                                             index */
022C 1BB6    BTFSC  36,7                  if( index < 0 )
022D 01B6    CLRF   36                     index = 0;
022E 3080    MOVLW  80h                   if( index > 88 )
022F 1283    BCF    STATUS,RP0
0230 0636    XORWF  36,W
0231 00A7    MOVWF  27
0232 30D8    MOVLW  D8h
0233 0227    SUBWF  27,W
0234 1D03    BTFSS  STATUS,Z
0235 1C03    BTFSS  STATUS,C
0236 2A3A    GOTO   023Ah
0237 3058    MOVLW  58h                    index = 88;
0238 1283    BCF    STATUS,RP0
0239 00B6    MOVWF  36

                                          /* Save predicted sample and quantizer step size
                                             index for next iteration */
023A 1283    BCF    STATUS,RP0            state.prevsample = (short)predsample;
023B 0832    MOVF   32,W
023C 00AB    MOVWF  2B
023D 01AC    CLRF   2C
023E 1BAB    BTFSC  2B,7
023F 03AC    DECF   2C
0240 1283    BCF    STATUS,RP0            state.previndex = index;
0241 0836    MOVF   36,W
0242 00AD    MOVWF  2D

                                          /* Return the new speech sample */
0243 0833    MOVF   33,W                  return( predsample );
0244 0084    MOVWF  FSR
0245 0832    MOVF   32,W
0246 0008    RETURN

                                      }
```

APPENDIX D: INTERACTIVE MULTIMEDIA ASSOCIATION INFORMATION

The IMA ADPCM Reference Algorithm is contained in the Digital Audio Doc-Pac. The Doc-Pac contains the following information:

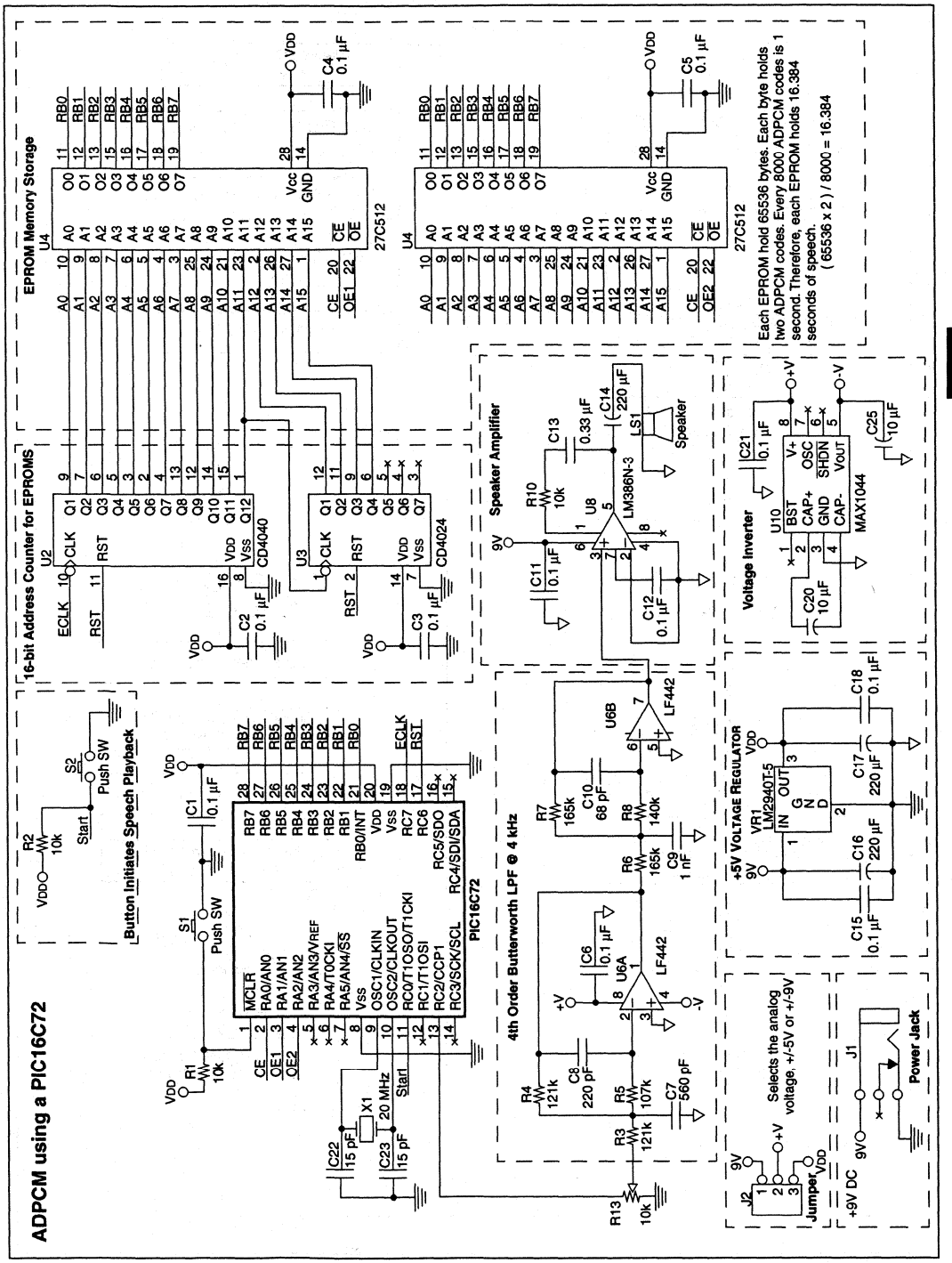
- *About IMA Digital Audio* - describes the IMA's activities in digital audio
- IMA *Recommended Practices for Enhancing Digital Audio Compatability in Multimedia Systems* (version 3.0). This document contains the official technical recommendations including the ADPCM algorithm
- *IMA Digital Audio Special Edition Proceedings* This document contains a detailed notes of the evaluation process, code fragments and testing methodologies.
- Floppy disk with code fragments.
- Contact information for companies supporting IMA ADPCM.

The IMA can be reached at:

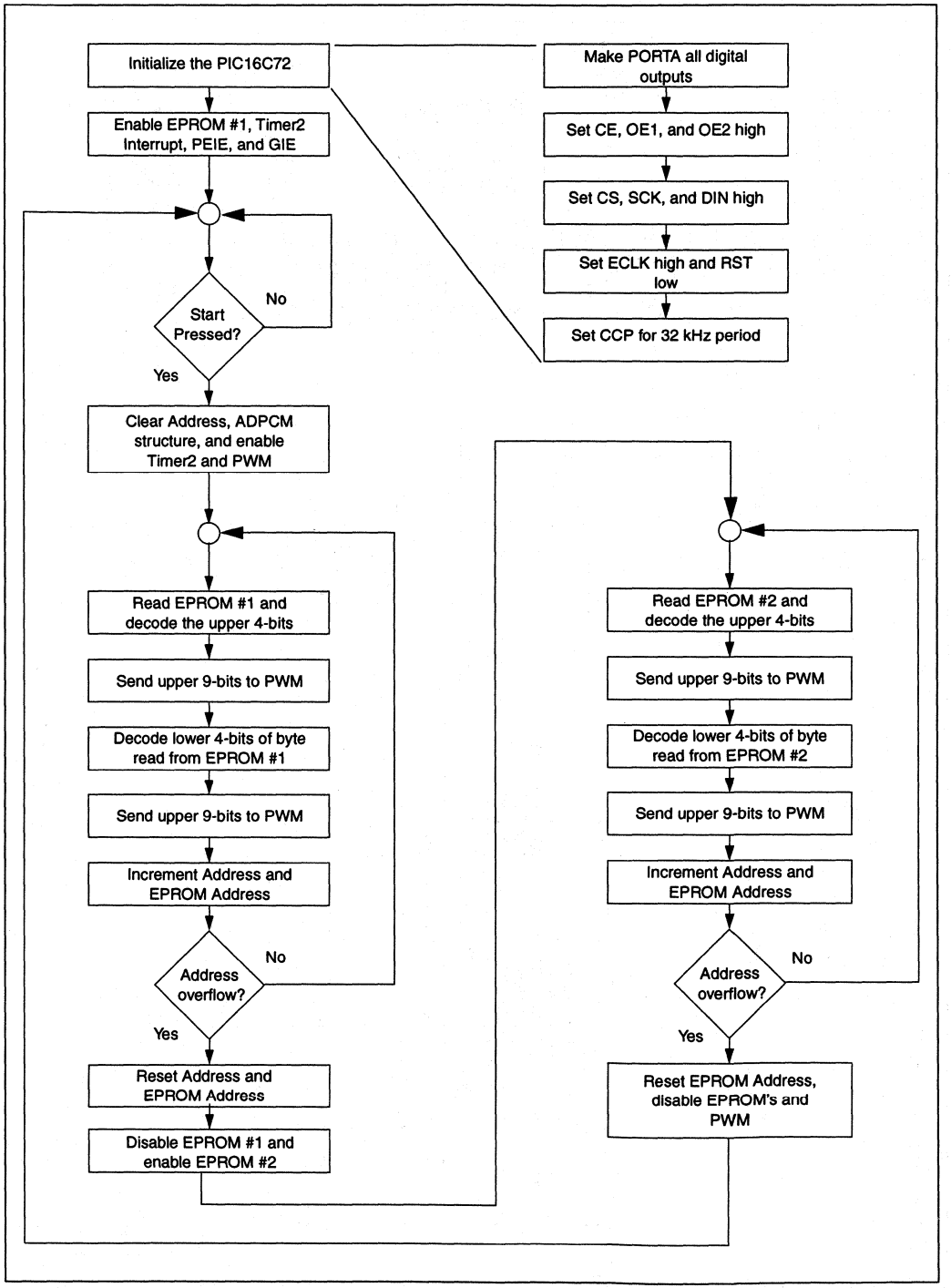
Interactive Multimedia Association
48 Maryland Avenue, Suite 202
Annapolis, MD 21401-8011 USA
Tel: (410)-626-1380
Fax: (410)-263-0590
http://www.ima.org

© 1997 Microchip Technology Inc.

APPENDIX E: APPLICATION HARDWARE SCHEMATIC



APPENDIX F: APPLICATION FIRMWARE BLOCK DIAGRAM AND LISTING



© 1997 Microchip Technology Inc.

Firmware Listing

```
/*****************************************************************************
*    Filename:    SPEECH.C                                                    *
*****************************************************************************
*    Author:      Rodger Richey                                               *
*    Title:       Senior Applications Engineer                                *
*    Company:     Microchip Technology Incorporated                           *
*    Revision:    0                                                           *
*    Date:        1-9-96                                                      *
*    Compiled using Bytecraft Ltd. MPC version BC.193                         *
*****************************************************************************
*    Include files:                                                           *
*       16c72.h - Standard include file for the PIC16C72 device               *
*       delay14.h - Standard include file for delay routines                  *
*       adpcm.h - Include file with ADPCM related information (Rev0)          *
*       adpcm.c - Source file with ADPCM routines (Rev0)                      *
*****************************************************************************
*    This file contains the routines to:                                      *
*       - Initialize the PIC16C72                                             *
*       - Read data from the EPROMs                                           *
*       - Call the ADPCM decoder routine                                      *
*       - Send data to the DAC and PWM                                        *
*       - Increment the address of the EPROMs                                 *
*****************************************************************************
*    External Clock Frequency: 20MHz                                          *
*       Configuration Bit Settings                                            *
*             OSC:     HS                                                     *
*             WDT:     Off                                                    *
*             PWRTE:   On                                                     *
*             CP:      Off                                                    *
*    Compression and Decompression (only Decompression is used)               *
*       Program Memory Usage:          906 words                              *
*       Data Memory Usage:             24 bytes                               *
*    Compression only                                                         *
*       Program Memory Usage:          686 words                              *
*       Data Memory Usage:             22 bytes                               *
*****************************************************************************/
                                #pragma option v
                                #include <16c72.h>
                                #pragma option +l;
0FFB                            #define MAXROM 2043
0005 0FFB                       #pragma memory ROM [MAXROM] @ 0x05;
0020 0060                       #pragma memory RAM [0x96]   @ 0x20;
00A0 0060                       #pragma memory RAM [0x32]   @ 0xA0;
                                #pragma option +l;

                                #include "adpcm.h"
/*****************************************************************************
*    Filename:    PCADPCM.H                                                   *
*****************************************************************************
*    Author:      Rodger Richey                                               *
*    Title:       Senior Applications Engineer                                *
*    Company:     Microchip Technology Incorporated                           *
*    Revision:    0                                                           *
*    Date:        1-11-96                                                     *
*    Compiled using Bytecraft Ltd. MPC version BC.193                         *
*****************************************************************************
*    This is the header file that contains the ADPCM structure definition     *
*    and the function prototypes.                                             *
*****************************************************************************/
                                struct ADPCMState {
                                   signed long prevsample;       / * Predicted sample */
                                   int         previndex; /*Index into step size table*/
                                };
```

```
                                /* Function prototype for the ADPCM Encoder routine */
                                signed long ADPCMDecoder( char );

                                /* Function prototype for the ADPCM Decoder routine */
                                char ADPCMEncoder( signed long );

                                // Defines for PORTA
0000                            #define CE   0   // Chip Enable for EPROMS
0001                            #define OE1  1   // Output Enable for EPROM #1
0002                            #define OE2  2   // Output Enable for EPROM #2

                                // Defines for PORTC
0000                            #define STRT 0   // Start Button input
0002                            #define PWM  2   // PWM output
0006                            #define RST  6   // EPROM counter reset
0007                            #define ECLK 7   // EPROM counter clock

0025                            unsigned long Address;  // EPROM address
002A                            unsigned char Wait;
002B 0003                       struct ADPCMState state;// ADPCM structure

                                #include "adpcm.c"
```

---Insert Appendix C here---

```
/*****************************************************************************
*    Init72 - This function initializes PORTA, PORTC, and PWM                *
*****************************************************************************
*    Input Variables:                                                        *
*       None                                                                 *
*    Return Variables:                                                       *
*       None                                                                 *
*****************************************************************************/
                                void Init72(void)
                                {
0247 3008    MOVLW  08h             OPTION = 8;
0248 1683    BSF    STATUS,RP0
0249 0081    MOVWF  TMR0
024A 3007    MOVLW  07h             ADCON1 = 7;     // Make PORTA all digital
024B 009F    MOVWF  ADCON0
024C 1283    BCF    STATUS,RP0      PORTA = 7;      // Set CE,OE1,OE2
024D 0085    MOVWF  PORTA
024E 1683    BSF    STATUS,RP0      TRISA = 0;      // Make PORTA all outputs
024F 0185    CLRF   PORTA
0250 3040    MOVLW  40h             PORTC = 0x40;   // Set ECLK, all outputs
0251 1283    BCF    STATUS,RP0
0252 0087    MOVWF  PORTC
0253 3005    MOVLW  05h             TRISC = 0x05;   // Clear RST output, STRT input
0254 1683    BSF    STATUS,RP0
0255 0087    MOVWF  PORTC
0256 309B    MOVLW  9Bh             PR2 = 155;      // Set PWM for 32kHz period
0257 0092    MOVWF  T2CON
0258 1283    BCF    STATUS,RP0      CCPR1L = 0;
0259 0195    CLRF   CCPR1L
025A 3018    MOVLW  18h             T2CON = 0x18;   // Set Timer2 postscaler to 4
025B 0092    MOVWF  T2CON
025C 0008    RETURN                 return;
                                }

/*****************************************************************************
*    main - This function controls everything, kinda like a god.             *
*****************************************************************************
*    Input Variables:                                                        *
```


© 1997 Microchip Technology Inc.

```
*       None                                                                  *
*    Return Variables:                                                        *
*       None                                                                  *
*************************************************************************/
                                    void main(void)
                                    {
003B                                    char code;              // EPROM byte
003C                                    signed long sample;     // decoded sample
003E                                    unsigned long pwmout;   // temporary variable for PWM

025D 2247    CALL   0247h               Init72();               // Initialize the PIC16C72

025E 3001    MOVLW  01h                 Wait = 1;
025F 1283    BCF    STATUS,RP0
0260 00AA    MOVWF  2A
0261 1707    BSF    PORTC,6             PORTC.RST = 1;          // Reset the EPROM counter
0262 0000    NOP                        NOP();
0263 0000    NOP                        NOP();
0264 1307    BCF    PORTC,6             PORTC.RST = 0;
0265 0000    NOP                        NOP();
0266 0000    NOP                        NOP();
0267 1085    BCF    PORTA,1             PORTA.OE1 = 0;          // Enable EPROM #1
0268 0000    NOP                        NOP();
0269 0000    NOP                        NOP();
026A 1005    BCF    PORTA,0             PORTA.CE = 0;
026B 0000    NOP                        NOP();
026C 0000    NOP                        NOP();
026D 108C    BCF    PIR1,1              PIR1.TMR2IF = 0;        // Enable Timer2 Interrupts
026E 1683    BSF    STATUS,RP0          PIE1.TMR2IE = 1;
026F 148C    BSF    PIR1,1
0270 170B    BSF    INTCON,PEIE         INTCON.PEIE = 1;        // Enable PEIE and GIE
0271 178B    BSF    INTCON,GIE          INTCON.GIE = 1;

                                        while(1)
                                        {
0272 1283    BCF    STATUS,RP0
0273 1807    BTFSC  PORTC,0
0274 2A72    GOTO   0272h
0275                                      while(PORTC.STRT); // Wait for Start button to be
0275 1283    BCF    STATUS,RP0            while(!PORTC.STRT); // pressed then released
0276 1C07    BTFSS  PORTC,0
0277 2A75    GOTO   0275h
0278 1283    BCF    STATUS,RP0            Address = 0;        // Clear EPROM address
0279 01A5    CLRF   25
027A 01A6    CLRF   26
027B 01AB    CLRF   2B                    state.prevsample = 0; // Clear ADPCM structure
027C 01AC    CLRF   2C
027D 01AD    CLRF   2D                    state.previndex = 0;

027E 1683    BSF    STATUS,RP0            TRISC.PWM = 0;      // Make PWM an output
027F 1107    BCF    PORTC,2
0280 1283    BCF    STATUS,RP0            T2CON.TMR2ON = 1;   // Enable Timer2
0281 1512    BSF    T2CON,2
0282 300F    MOVLW  0Fh                   CCP1CON = 0x0f;     // Enable PWM
0283 0097    MOVWF  CCP1CON
                                          do                  // Decode upper 4 bits of code
                                          {
0284 1283    BCF    STATUS,RP0              code = PORTB;
0285 0806    MOVF   PORTB,W
0286 00BB    MOVWF  3B
0287 083B    MOVF   3B,W                    sample = ADPCMDecoder( (code>>4)&0x0f );
0288 00A7    MOVWF  27
0289 0EA7    SWAPF  27
028A 300F    MOVLW  0Fh
028B 0527    ANDWF  27,W
028C 390F    ANDLW  0Fh
```

```
028D 21A5    CALL   01A5h
028E 1283    BCF    STATUS,RP0
028F 00BC    MOVWF  3C
0290 0804    MOVF   FSR,W
0291 00BD    MOVWF  3D
0292 1283    BCF    STATUS,RP0          while(Wait);      // Wait for 8kHz to output
0293 082A    MOVF   2A,W
0294 3800    IORLW  00h
0295 1D03    BTFSS  STATUS,Z
0296 2A92    GOTO   0292h
0297 3001    MOVLW  01h                 Wait = 1;
0298 1283    BCF    STATUS,RP0
0299 00AA    MOVWF  2A
029A 083C    MOVF   3C,W                pwmout = sample; // Write to PWM
029B 00BE    MOVWF  3E
029C 083D    MOVF   3D,W
029D 00BF    MOVWF  3F
029E 1FBD    BTFSS  3D,7                if(sample<0)      // Add offset to
029F 2AA4    GOTO   02A4h
02A0 3080    MOVLW  80h                  pwmout -= 0x8000; // sample
02A1 023F    SUBWF  3F,W
02A2 00BF    MOVWF  3F
02A3 2AA6    GOTO   02A6h               else
02A4 3080    MOVLW  80h                  pwmout += 0x8000;
02A5 07BF    ADDWF  3F
02A6 083F    MOVF   3F,W                CCPR1L = (pwmout>>9)&0x007f; // Write 7 bits
02A7 00A7    MOVWF  27
02A8 0CA7    RRF    27
02A9 307F    MOVLW  7Fh
02AA 0527    ANDWF  27,W
02AB 0095    MOVWF  CCPR1L
02AC 30CF    MOVLW  CFh                 CCP1CON &= 0xcf;              // to CCPR1L
02AD 0597    ANDWF  CCP1CON
02AE 183F    BTFSC  3F,0                if(pwmout&0x0100)             // Write 2 bits
02AF 1697    BSF    CCP1CON,5            CCP1CON.5 = 1;              // to CCP1CON
02B0 1283    BCF    STATUS,RP0          if(pwmout&0x0080)             // to get 9-bit
02B1 1BBE    BTFSC  3E,7
02B2 1617    BSF    CCP1CON,4            CCP1CON.4 = 1;              // PWM

                                                    // Decode lower 4 bits of code
02B3 300F    MOVLW  0Fh                 sample = ADPCMDecoder(code&0x0f);
02B4 1283    BCF    STATUS,RP0
02B5 053B    ANDWF  3B,W
02B6 118A    BCF    PCLATH,3
02B7 21A5    CALL   01A5h
02B8 118A    BCF    PCLATH,3
02B9 1283    BCF    STATUS,RP0
02BA 00BC    MOVWF  3C
02BB 0804    MOVF   FSR,W
02BC 00BD    MOVWF  3D
02BD 1283    BCF    STATUS,RP0          while(Wait);      // Wait for 8kHz to output
02BE 082A    MOVF   2A,W
02BF 3800    IORLW  00h
02C0 1D03    BTFSS  STATUS,Z
02C1 2ABD    GOTO   02BDh
02C2 3001    MOVLW  01h                 Wait = 1;
02C3 1283    BCF    STATUS,RP0
02C4 00AA    MOVWF  2A
02C5 083C    MOVF   3C,W                pwmout = sample; // Write to PWM
02C6 00BE    MOVWF  3E
02C7 083D    MOVF   3D,W
02C8 00BF    MOVWF  3F
02C9 1FBD    BTFSS  3D,7                if(sample<0)      // Add offset to
02CA 2ACF    GOTO   02CFh
02CB 3080    MOVLW  80h                  pwmout -= 0x8000; // sample
02CC 023F    SUBWF  3F,W
```


© 1997 Microchip Technology Inc.

```
02CD 00BF    MOVWF  3F
02CE 2AD1    GOTO   02D1h                 else
02CF 3080    MOVLW  80h                   pwmout += 0x8000;
02D0 07BF    ADDWF  3F
02D1 083F    MOVF   3F,W                 CCPR1L = (pwmout>>9)&0x007f; // Write 7 bits
02D2 00A7    MOVWF  27
02D3 0CA7    RRF    27
02D4 307F    MOVLW  7Fh
02D5 0527    ANDWF  27,W
02D6 0095    MOVWF  CCPR1L
02D7 30CF    MOVLW  CFh                  CCP1CON &= 0xcf;              // to CCPR1L
02D8 0597    ANDWF  CCP1CON
02D9 183F    BTFSC  3F,0                 if(pwmout&0x0100)             // Write 2 bits
02DA 1697    BSF    CCP1CON,5             CCP1CON.5 = 1;               // to CCP1CON
02DB 1283    BCF    STATUS,RP0           if(pwmout&0x0080)             // to get 9-bit
02DC 1BBE    BTFSC  3E,7
02DD 1617    BSF    CCP1CON,4             CCP1CON.4 = 1;               // PWM

                                                        // Increment Address of EPROMS
02DE 1283    BCF    STATUS,RP0           Address++;
02DF 0AA5    INCF   25
02E0 1903    BTFSC  STATUS,Z
02E1 0AA6    INCF   26
02E2 1787    BSF    PORTC,7              PORTC.ECLK = 1;
02E3 0000    NOP                         NOP();
02E4 0000    NOP                         NOP();
02E5 1387    BCF    PORTC,7              PORTC.ECLK = 0;
02E6 0000    NOP                         NOP();
02E7 0000    NOP                         NOP();
02E8 0826    MOVF   26,W                } while(Address);  // Has Address overflowed?
02E9 0425    IORWF  25,W
02EA 118A    BCF    PCLATH,3
02EB 1D03    BTFSS  STATUS,Z
02EC 2A84    GOTO   0284h

02ED 1283    BCF    STATUS,RP0           PORTA.CE = 1;       // Disable EPROM #1
02EE 1405    BSF    PORTA,0
02EF 0000    NOP                        NOP();
02F0 0000    NOP                        NOP();
02F1 1485    BSF    PORTA,1             PORTA.OE1 = 1;
02F2 0000    NOP                        NOP();
02F3 0000    NOP                        NOP();
02F4 1705    BSF    PORTA,6             PORTA.RST = 1;      // Reset EPROM counter
02F5 0000    NOP                        NOP();
02F6 0000    NOP                        NOP();
02F7 1305    BCF    PORTA,6             PORTA.RST = 0;
02F8 0000    NOP                        NOP();
02F9 0000    NOP                        NOP();
02FA 1105    BCF    PORTA,2             PORTA.OE2 = 0;      // Enable EPROM #2
02FB 0000    NOP                        NOP();
02FC 0000    NOP                        NOP();
02FD 1005    BCF    PORTA,0             PORTA.CE = 0;
02FE 0000    NOP                        NOP();
02FF 0000    NOP                        NOP();
0300 01A5    CLRF   25                  Address = 0;        // Clear EPROM Address
0301 01A6    CLRF   26
0302 0000    NOP                        NOP();
0303 0000    NOP                        NOP();

                                        do
                                        {                   // Decode upper 4 bits of code
0304 1283    BCF    STATUS,RP0            code = PORTB;
0305 0806    MOVF   PORTB,W
0306 00BB    MOVWF  3B
0307 083B    MOVF   3B,W                  sample = ADPCMDecoder( (code>>4)&0x0f );
0308 00A7    MOVWF  27
```

```
0309 0EA7    SWAPF  27
030A 300F    MOVLW  0Fh
030B 0527    ANDWF  27,W
030C 390F    ANDLW  0Fh
030D 21A5    CALL   01A5h
030E 1283    BCF    STATUS,RP0
030F 00BC    MOVWF  3C
0310 0804    MOVF   FSR,W
0311 00BD    MOVWF  3D
0312 1283    BCF    STATUS,RP0          while(Wait);      // Wait for 8kHz
0313 082A    MOVF   2A,W
0314 3800    IORLW  00h
0315 1D03    BTFSS  STATUS,Z
0316 2B12    GOTO   0312h
0317 3001    MOVLW  01h                 Wait = 1;
0318 1283    BCF    STATUS,RP0
0319 00AA    MOVWF  2A
031A 083C    MOVF   3C,W                pwmout = sample; // Write to PWM
031B 00BE    MOVWF  3E
031C 083D    MOVF   3D,W
031D 00BF    MOVWF  3F
031E 1FBD    BTFSS  3D,7                if(sample<0)      // Add offset to
031F 2B24    GOTO   0324h
0320 3080    MOVLW  80h                  pwmout -= 0x8000; // sample
0321 023F    SUBWF  3F,W
0322 00BF    MOVWF  3F
0323 2B26    GOTO   0326h               else
0324 3080    MOVLW  80h                  pwmout += 0x8000;
0325 07BF    ADDWF  3F
0326 083F    MOVF   3F,W                CCPR1L = (pwmout>>9)&0x007f; // Write 7 bits
0327 00A7    MOVWF  27
0328 0CA7    RRF    27
0329 307F    MOVLW  7Fh
032A 0527    ANDWF  27,W
032B 0095    MOVWF  CCPR1L
032C 30CF    MOVLW  CFh                 CCP1CON &= 0xcf;              // to CCPR1L
032D 0597    ANDWF  CCP1CON
032E 183F    BTFSC  3F,0                if(pwmout&0x0100)             // Write 2 bits
032F 1697    BSF    CCP1CON,5            CCP1CON.5 = 1;              // to CCP1CON
0330 1283    BCF    STATUS,RP0          if(pwmout&0x0080)             // to get 9-bit
0331 1BBE    BTFSC  3E,7
0332 1617    BSF    CCP1CON,4            CCP1CON.4 = 1;              // PWM
                                                  // Decode lower 4 bits of code
0333 300F    MOVLW  0Fh                 sample = ADPCMDecoder(code&0x0f);
0334 1283    BCF    STATUS,RP0
0335 053B    ANDWF  3B,W
0336 118A    BCF    PCLATH,3
0337 21A5    CALL   01A5h
0338 118A    BCF    PCLATH,3
0339 1283    BCF    STATUS,RP0
033A 00BC    MOVWF  3C
033B 0804    MOVF   FSR,W
033C 00BD    MOVWF  3D
033D 1283    BCF    STATUS,RP0          while(Wait);      // Wait for 8kHz
033E 082A    MOVF   2A,W
033F 3800    IORLW  00h
0340 1D03    BTFSS  STATUS,Z
0341 2B3D    GOTO   033Dh
0342 3001    MOVLW  01h                 Wait = 1;
0343 1283    BCF    STATUS,RP0
0344 00AA    MOVWF  2A
0345 083C    MOVF   3C,W                pwmout = sample; // Write to PWM
0346 00BE    MOVWF  3E
0347 083D    MOVF   3D,W
0348 00BF    MOVWF  3F
0349 1FBD    BTFSS  3D,7                if(sample<0)      // Add offset to
```


© 1997 Microchip Technology Inc.

```
034A 2B4F    GOTO   034Fh
034B 3080    MOVLW  80h                    pwmout -= 0x8000; // sample
034C 023F    SUBWF  3F,W
034D 00BF    MOVWF  3F
034E 2B51    GOTO   0351h                  else
034F 3080    MOVLW  80h                    pwmout += 0x8000;
0350 07BF    ADDWF  3F
0351 083F    MOVF   3F,W                   CCPR1L = (pwmout>>9)&0x007f; // Write 7 bits
0352 00A7    MOVWF  27
0353 0CA7    RRF    27
0354 307F    MOVLW  7Fh
0355 0527    ANDWF  27,W
0356 0095    MOVWF  CCPR1L
0357 30CF    MOVLW  CFh                    CCP1CON &= 0xcf;              // to CCPR1L
0358 0597    ANDWF  CCP1CON
0359 183F    BTFSC  3F,0                   if(pwmout&0x0100)             // Write 2 bits
035A 1697    BSF    CCP1CON,5               CCP1CON.5 = 1;               // to CCP1CON
035B 1283    BCF    STATUS,RP0             if(pwmout&0x0080)             // to get 9-bit
035C 1BBE    BTFSC  3E,7
035D 1617    BSF    CCP1CON,4               CCP1CON.4 = 1;               // PWM

                                                          // Increment Address of EPROMS
035E 1283    BCF    STATUS,RP0             Address++;
035F 0AA5    INCF   25
0360 1903    BTFSC  STATUS,Z
0361 0AA6    INCF   26
0362 1787    BSF    PORTC,7                PORTC.ECLK = 1;
0363 0000    NOP                           NOP();
0364 0000    NOP                           NOP();
0365 1387    BCF    PORTC,7                PORTC.ECLK = 0;
0366 0000    NOP                           NOP();
0367 0000    NOP                           NOP();
0368 0826    MOVF   26,W                  } while(Address);  // Has Address overflowed?
0369 0425    IORWF  25,W
036A 118A    BCF    PCLATH,3
036B 1D03    BTFSS  STATUS,Z
036C 2B04    GOTO   0304h
036D 1283    BCF    STATUS,RP0           T2CON.TMR2ON = 0;  // Disable Timer2
036E 1112    BCF    T2CON,2
036F 0197    CLRF   CCP1CON              CCP1CON = 0;       // Disable PWM
0370 1683    BSF    STATUS,RP0           TRISC.PWM = 1;     // Make PWM input
0371 1507    BSF    PORTC,2
0372 1283    BCF    STATUS,RP0           PORTA.CE = 1;      // Disable EPROM #2
0373 1405    BSF    PORTA,0
0374 0000    NOP                         NOP();
0375 0000    NOP                         NOP();
0376 1505    BSF    PORTA,2              PORTA.OE2 = 1;
0377 0000    NOP                         NOP();
0378 0000    NOP                         NOP();
0379 1705    BSF    PORTA,6              PORTA.RST = 1;     // Reset EPROM counter
037A 0000    NOP                         NOP();
037B 0000    NOP                         NOP();
037C 1305    BCF    PORTA,6              PORTA.RST = 0;
037D 0000    NOP                         NOP();
037E 0000    NOP                         NOP();
037F 1085    BCF    PORTA,1              PORTA.OE1 = 0;     // Enable EPROM #1
0380 0000    NOP                         NOP();
0381 0000    NOP                         NOP();
0382 1005    BCF    PORTA,0              PORTA.CE = 0;
0383 2A72    GOTO   0272h             }
0384 0008    RETURN                }

/*************************************************************************
*    __INT - This is the interrupt service routine. Only Timer 2 overflow    *
*            is implemented.                                                 *
*************************************************************************
```

```
*    Input Variables:                                                          *
*       None                                                                   *
*    Return Variables:                                                         *
*       None                                                                   *
*****************************************************************************/
0004 2B85    GOTO   0385h          void __INT(void)
0385                               {
0385 1283    BCF    STATUS,RP0         if(PIR1.TMR2IF)        // Timer2 overflow interrupt
0386 1C8C    BTFSS  PIR1,1
0387 2B8A    GOTO   038Ah
0388                                    {
0388 01AA    CLRF   2A                     Wait = 0;
0389 108C    BCF    PIR1,1                 PIR1.TMR2IF = 0;   // Clear flag
                                        }
038A 0009    RETFIE                     return;
                                   }
ROM USAGE MAP

    0000 to 0002    0004 to 038A
    Total ROM used 038A

Errors             :    0
Warnings           :    0
```


© 1997 Microchip Technology Inc.

APPENDIX G: PC ADPCM ENCODER/DECODER PROGRAM

PCSPEECH.C

```
/*****************************************************************************
*       Filename:  PCSPEECH.C                                                *
*****************************************************************************
*       Author:          Rodger Richey                                       *
*       Title:          Senior Applications Engineer                         *
*       Company:         Microchip Technology Incorporated                   *
*       Revision:        0                                                   *
*       Date:           1-11-96                                              *
*       Compiled using Borland C+ Version 3.1                                *
*****************************************************************************
*       Include files:                                                       *
*               stdio.h - Standard input/output header file                  *
*               stdlib.h - Standard library header file                      *
*               string.h - Standard string header file                       *
*               pcadpcm.h - ADPCM related information header file (Rev0)     *
*****************************************************************************
*       Usage:                                                               *
*               ADPCM Encode - pcspeech e <infile> <outfile>                 *
*                              <infile> is a 16-bit raw speech file          *
*                              <outfile> contains the ADPCM codes            *
*               ADPCM Decode - pcspeech d <infile> <outfile>                 *
*                              <infile> contains the ADPCM codes             *
*                              <outfile> is a 16-bit raw speech file         *
*****************************************************************************
*       This file contains the code to:                                      *
*               - Open the input and output files                            *
*               - Read data from the input file                              *
*               - Call the encode/decode routines                            *
*               - Write data to the output file                              *
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pcadpcm.h"

/*****************************************************************************
*       Usage - this routine prints a how to message for the pcspeech prgm   *
*****************************************************************************
*       Input variables:                                                     *
*               None                                                         *
*       Output variables:                                                    *
*               None                                                         *
*****************************************************************************/
void Usage(void)
{
    printf("ADPCM Encoder/Decoder -- usage:\n");
    printf("\tEncoder = pcspeech e infile outfile\n");
    printf("\tDecoder = pcspeech d infile outfile\n");
    exit(1);
}

/*****************************************************************************
*       main - controls file I/O and ADPCM calls                             *
*****************************************************************************
*       Input variables:                                                     *
*               int argc - number of arguements in argv                      *
*               char **argv - pointer to an array of strings                 *
*       Return variables:                                                    *
*               None                                                         *
*****************************************************************************/
void main(
    int   argc,
    char  **argv)
```


© 1997 Microchip Technology Inc.

```
{
    int         which;
    short                   sample;
    unsigned char           code;
    int                     n;
    struct ADPCMstate       state;
    FILE        *fpin;
    FILE        *fpout;

    state.prevsample=0;
    state.previndex=0;

    /* Determine if this is an encode or decode operation */
    if(argc <= 1)
        Usage();
    else if( strcmp(argv[1],"e")==0 || strcmp(argv[1],"E")==0 )
        which = 0;
    else if( strcmp(argv[1],"d")==0 || strcmp(argv[1],"D")==0 )
        which = 1;
    argc--;
    argv++;

    /* Open input file for processing */
    if(argc <= 1)
        Usage();
    else if( (fpin=fopen(argv[1],"rb"))==NULL)
    {
        printf("ADPCM Encoder/Decoder\n");
        printf("ERROR: Could not open %s for input\n",argv[1]);
        exit(1);
    }
    argc--;
    argv++;

    /* Open output file */
    if(argc <= 1)
    {
        fclose(fpin);
        Usage();
    }
    else if( (fpout=fopen(argv[1],"wb"))==NULL)
    {
        fclose(fpin);
        printf("ADPCM Encoder/Decoder\n");
        printf("ERROR: Could not open %s for output\n",argv[1]);
        exit(1);
    }

    // ADPCM Decoder selected
    if(which)
    {
        printf("ADPCM Decoding in progress\n");
        /* Read and unpack input codes and process them */
        while (fread(&code, sizeof (char), 1, fpin) == 1)
        {
            // Send the upper 4-bits of code to decoder
            sample = ADPCMDecoder((code>>4)&0x0f, &state);
            // Write sample for upper 4-bits of code
            fwrite(&sample, sizeof(short), 1, fpout);

            // Send the lower 4-bits of code to decoder
            sample = ADPCMDecoder(code&0x0f,&state);
            // Write sample for lower 4-bits of code
            fwrite(&sample,sizeof(short),1,fpout);
        }
    }
```


© 1997 Microchip Technology Inc.

```
    // ADPCM Encoder selected
    else
    {
        printf("ADPCM Encoding in progress\n");
        /* Read input file and process */
        while (fread(&sample, sizeof(short), 1, fpin) == 1)
        {
            // Encode sample into lower 4-bits of code
            code = ADPCMEncoder(sample,&state);
            // Move ADPCM code to upper 4-bits
            code = (code << 4) & 0xf0;
            // Read new sample from file
            if(fread(&sample,sizeof(short),1,fpin)==0)
            {
                // No more samples, write code to file
                fwrite(&code,sizeof(char),1,fpout);
                break;
            }
            // Encode sample and save in lower 4-bits of code
            code |= ADPCMEncoder(sample,&state);
            // Write code to file, code contains 2 ADPCM codes
            fwrite(&code, sizeof (char), 1, fpout);
        }
    }

    fclose(fpin);
    fclose(fpout);
}
```

PCADPCM.H

```
/*****************************************************************************
*       Filename:  PCADPCM.H                                                 *
*****************************************************************************
*       Author:          Rodger Richey                                       *
*       Title:          Senior Applications Engineer                         *
*       Company:         Microchip Technology Incorporated                   *
*       Revision:        0                                                   *
*       Date:           1-11-96                                              *
*       Compiled using Borland C+ Version 3.1                                *
*****************************************************************************
*       This is the header file that contains the ADPCM structure definition *
*       and the function prototypes.                                         *
*****************************************************************************/
struct ADPCMstate {
    short prevsample;/* Predicted sample */
    int previndex;/* Index into step size table */
};

/* Function prototype for the ADPCM Encoder routine */
char ADPCMEncoder(short , struct ADPCMstate *);

/* Function prototype for the ADPCM Decoder routine */
int ADPCMDecoder(char , struct ADPCMstate *);
```

PCADPCM.C

```
/*****************************************************************************
*       Filename:  PCADPCM.C                                                 *
*****************************************************************************
*       Author:          Rodger Richey                                       *
*       Title:          Senior Applications Engineer                         *
*       Company:         Microchip Technology Incorporated                   *
*       Revision:        0                                                   *
*       Date:           1-11-96                                              *
*       Compiled using Borland C+ Version 3.1                                *
```

```
*****************************************************************************
*       Include files:                                                      *
*               stdio.h - Standard input/output header file                 *
*               pcadpcm.h - ADPCM related information header file (Rev0)    *
*****************************************************************************
*       This file contains the ADPCM encode and decode routines.  These     *
*       routines were obtained from the Interactive Multimedia Association's *
*       Reference ADPCM algorithm.  This algorithm was first implemented by  *
*       Intel/DVI.                                                          *
*****************************************************************************/

#include <stdio.h>
#include "pcadpcm.h"

/* Table of index changes */
signed char IndexTable[16] = {
    -1, -1, -1, -1, 2, 4, 6, 8,
    -1, -1, -1, -1, 2, 4, 6, 8,
};

/* Quantizer step size lookup table */
int StepSizeTable[89] = {
    7, 8, 9, 10, 11, 12, 13, 14, 16, 17,
    19, 21, 23, 25, 28, 31, 34, 37, 41, 45,
    50, 55, 60, 66, 73, 80, 88, 97, 107, 118,
    130, 143, 157, 173, 190, 209, 230, 253, 279, 307,
    337, 371, 408, 449, 494, 544, 598, 658, 724, 796,
    876, 963, 1060, 1166, 1282, 1411, 1552, 1707, 1878, 2066,
    2272, 2499, 2749, 3024, 3327, 3660, 4026, 4428, 4871, 5358,
    5894, 6484, 7132, 7845, 8630, 9493, 10442, 11487, 12635, 13899,
    15289, 16818, 18500, 20350, 22385, 24623, 27086, 29794, 32767
};

/*****************************************************************************
*       ADPCMEncoder - ADPCM encoder routine                                 *
*****************************************************************************
*       Input variables:                                                    *
*               short sample - 16-bit signed speech sample                  *
*               struct ADPCMstate *state - ADPCM structure                  *
*       Return variables:                                                   *
*               char - 8-bit number containing the 4-bit ADPCM code         *
*****************************************************************************/
char ADPCMEncoder( short sample , struct ADPCMstate *state )
{
    int code;/* ADPCM output value */
    int diff;/* Difference between sample and the predicted
                                    sample */
    int step;/* Quantizer step size */
    int predsample;/* Output of ADPCM predictor */
    int diffq;/* Dequantized predicted difference */
    int index;/* Index into step size table */

    /* Restore previous values of predicted sample and quantizer step
       size index
    */
    predsample = (int)(state->prevsample);
    index = state->previndex;
    step = StepSizeTable[index];


    /* Compute the difference between the acutal sample (sample) and the
       the predicted sample (predsample)
    */
    diff = sample - predsample;
    if(diff >= 0)
        code = 0;
```


© 1997 Microchip Technology Inc.

```
else
{
    code = 8;
    diff = -diff;
}

/* Quantize the difference into the 4-bit ADPCM code using the
   the quantizer step size
*/
/* Inverse quantize the ADPCM code into a predicted difference
   using the quantizer step size
*/
diffq = step >> 3;
if( diff >= step )
{
    code |= 4;
    diff -= step;
    diffq += step;
}
step >>= 1;
if( diff >= step )
{
    code |= 2;
    diff -= step;
    diffq += step;
}
step >>= 1;
if( diff >= step )
{
    code |= 1;
    diffq += step;
}

/* Fixed predictor computes new predicted sample by adding the
   old predicted sample to predicted difference
*/
if( code & 8 )
    predsample -= diffq;
else
    predsample += diffq;

/* Check for overflow of the new predicted sample
*/
if( predsample > 32767 )
    predsample = 32767;
else if( predsample < -32767 )
    predsample = -32767;

/* Find new quantizer stepsize index by adding the old index
   to a table lookup using the ADPCM code
*/
index += IndexTable[code];

/* Check for overflow of the new quantizer step size index
*/
if( index < 0 )
    index = 0;
if( index > 88 )
    index = 88;

/* Save the predicted sample and quantizer step size index for
   next iteration
*/
state->prevsample = (short)predsample;
state->previndex = index;
```

© 1997 Microchip Technology Inc.

```
    /* Return the new ADPCM code
    */
    return ( code & 0x0f );
}

/*******************************************************************************
*       ADPCMDecoder - ADPCM decoder routine                                   *
*******************************************************************************
*       Input variables:                                                       *
*               char code - 8-bit number containing the 4-bit ADPCM code       *
*               struct ADPCMstate *state - ADPCM structure                     *
*       Return variables:                                                      *
*               int - 16-bit signed speech sample                              *
*******************************************************************************/
int ADPCMDecoder(char code, struct ADPCMstate *state)
{
    int step;/* Quantizer step size */
    int predsample;/* Output of ADPCM predictor */
    int diffq;/* Dequantized predicted difference */
    int index;/* Index into step size table */

    /* Restore previous values of predicted sample and quantizer step
       size index
    */
    predsample = (int)(state->prevsample);
    index = state->previndex;

    /* Find quantizer step size from lookup table using index
    */
    step = StepSizeTable[index];

    /* Inverse quantize the ADPCM code into a difference using the
       quantizer step size
    */
    diffq = step >> 3;
    if( code & 4 )
        diffq += step;
    if( code & 2 )
        diffq += step >> 1;
    if( code & 1 )
        diffq += step >> 2;

    /* Add the difference to the predicted sample
    */
    if( code & 8 )
        predsample -= diffq;
    else
        predsample += diffq;

    /* Check for overflow of the new predicted sample
    */
    if( predsample > 32767 )
        predsample = 32767;
    else if( predsample < -32767 )
        predsample = -32767;

    /* Find new quantizer step size by adding the old index and a
       table lookup using the ADPCM code
    */
    index += IndexTable[code];

    /* Check for overflow of the new quantizer step size index
    */
    if( index < 0 )
        index = 0;
    if( index > 88 )
```


© 1997 Microchip Technology Inc.

```
        index = 88;


    /* Save predicted sample and quantizer step size index for next
       iteration
    */
    state->prevsample = (short)predsample;
    state->previndex = index;

    /* Return the new speech sample */
    return( predsample );

}
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Yet Another Clock Featuring the PIC16C924

Author: Rodger Richey
Microchip Technology Inc.

INTRODUCTION

Once again, because of its universal familiarity and range of functionality, the clock is used to convey the use of the PIC16C92X microcontrollers. In this case we have added a twist to the clock with the addition of a thermometer. The LCD panel has a two digit temperature readout.

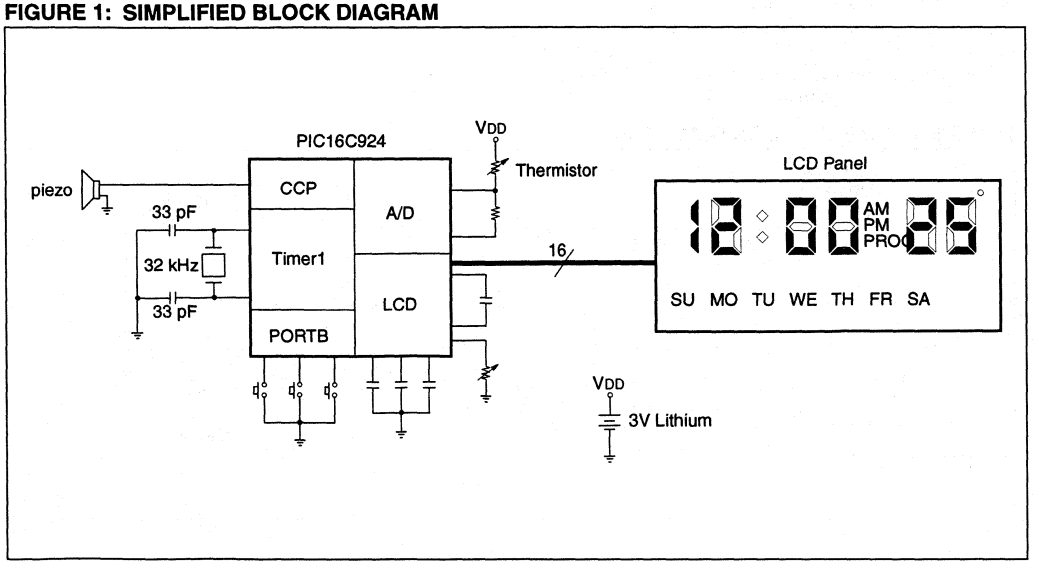
This application note will discuss the use of the following peripherals used to implement the clock: Timer1, PORTB, CCP, A/D converter, and the LCD Module. All source code and examples are written in C and compiled using Microchip's MPLAB-C compiler.

The features of the PIC16C924 are:

- 4K x 14 EPROM program memory
- 176 x 8 SRAM data memory
- DC - 8 MHz operating speed
- Timer0, Timer1, and Timer2
- One CCP pin
- SSP Module with SPI and I^2C capability
- 8-bit, 5 channel A/D converter
- LCD Module
  - Multiple LCD timing sources
  - LCD can be driven while in SLEEP
  - Static, 1/2, 1/3 and 1/4 multiplex modes
  - Static and 1/3 bias capability
  - Up to 32 segments, up to 4 commons
    - 1 COM x 32 SEGs = 32 pixels
    - 2 COMs x 31 SEGs = 62 pixels
    - 3 COMs x 30 SEGs = 90 pixels
    - 4 COMs x 29 SEGs = 116 pixels
- Available in DIE form, 68-pin PLCC, and 64-pin TQFP packages

3

FIGURE 1: SIMPLIFIED BLOCK DIAGRAM



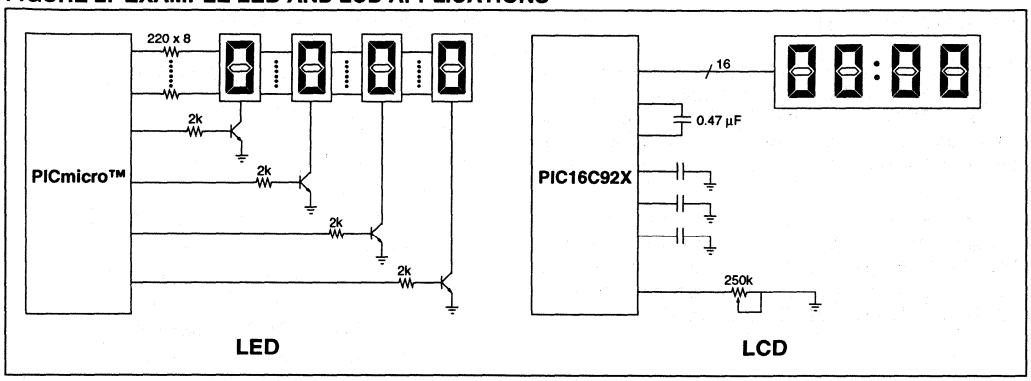
© 1997 Microchip Technology Inc.

LCD panels offer many advantages over LED type displays such as; lower cost, lower power consumption, and better display quality. Figure 2 shows typical examples of an LED and an LCD application. Table 1 further describes each application according to components, cost, power consumption, etc.

TABLE 1: LED vs. LCD

	LED	LCD
Cost (1000 units)	$7.05	$5.42
# Components	20	6
Power Consumption	~ 10 mA	~ 50 µA
Hardware	Timer, 12 I/O pins	LCD Module
Signal Generation	Firmware	LCD Module

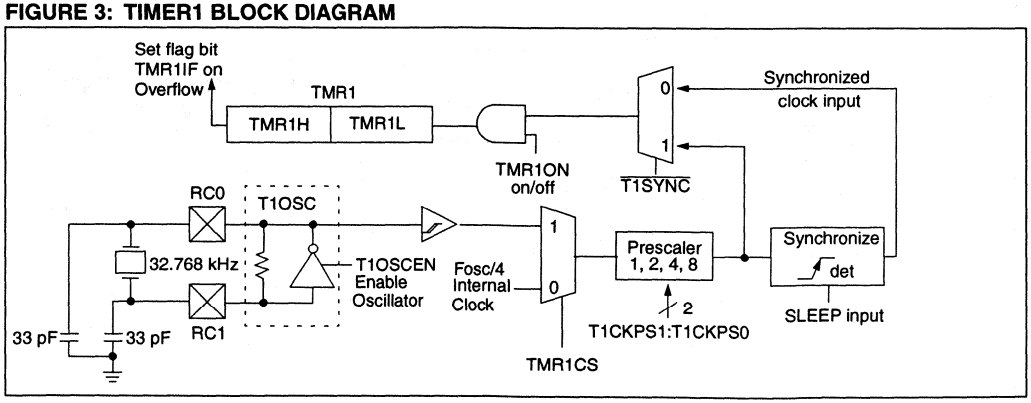
FIGURE 2: EXAMPLE LED AND LCD APPLICATIONS



TIMER1

Currently, the Timer1 module exists in all the PIC16CXXX devices with 28 or more pins. This module can be used to easily implement a real-time clock. Instead of an external real-time clock device, an inexpensive 32.768 kHz watch crystal and two 33 pF capacitors are used to complete the circuit. Figure 3 shows the block diagram for Timer1.

FIGURE 3: TIMER1 BLOCK DIAGRAM



© 1997 Microchip Technology Inc.

In this application, Timer1 is clocked by an external crystal connected across RC0 and RC1. The following is some sample code to initialize Timer1.

```
TMR1H = 0x80;
TMR1L = 0x00;
T1CON = 0b00001111;
PIR1.TMR1IF = 0;
PIE1.TMR1IE = 1;
```

The first step in initializing Timer1 is to preset TMR1H:TMR1L to 0x8000. Since Timer1 is a 16-bit timer, the 32.768 kHz crystal will cause Timer1 to overflow every two seconds or 65536 counts. For this reason, Timer1 is initialized to 0x8000 so that every overflow relates to one second. The second step is to configure Timer1. For this mode, the Timer1 oscillator must be enabled via T1CON<T1OSCEN>. This bit enables the internal oscillator, which is functionally equivalent to the LP oscillator of the microcontroller. The clock source for Timer1 is selected to be the external input using the T1CON<TMR1CS> bit. The prescaler is set to 1:1 using the T1CKPS1:T1CKPS0 bits of the T1CON register. Since the timer is to operate during SLEEP, it is not synchronized to the internal Fosc clock. Synchronization is controlled by the $\overline{\text{T1SYNC}}$ bit of the T1CON register. Finally, the clock source is enabled to clock TMR1H:TMR1L using the TMR1ON bit of T1CON.

In this mode of operation, Timer1 can operate during SLEEP while consuming a minimal amount of current (≈20 µA). The last two lines of code clear the Timer1 Overflow flag and enable the Timer1 Overflow interrupt. This interrupt will wake up the processor from SLEEP at a predetermined rate for updating the clock, in this case once each second. The following is a sample interrupt service routine for Timer1 Overflow:

```
if(PIR1.TMR1IF)
{
   Seconds++;              //Increment seconds
   if(Seconds > 59)        //60 seconds?
   {
      Seconds = 0;
      Minutes++;
   }
   if(Minutes > 59)        //60 minutes?
   {
      Minutes = 0;
      Hours++;
   }
   if(Hours > 12)          //Do not use 24hr
      Hours = 1;           //military time

   TMR1H |= 0x80;          //Reset timer1
   PIR1.TMR1IF = 0;        //Clear flag
}
```

First, register PIR1 is checked to verify that the Timer1 Overflow interrupt occurred. Next the variable `Seconds` is incremented every time a Timer1 overflow occurs. Once `Seconds` reaches 60, the `Minutes` variable is incremented and `Seconds` is cleared thus implementing 60 seconds per minute. If `Minutes` reaches 60, then the `Hours` variable is incremented and `Minutes` is cleared which implements 60 minutes per hour. For civilian time, if `Hours` is greater than 12, `Hours` is reset to 1. Military time uses 24 hours a day, so that when `Hours` reaches 24 it is reset to 1.

Since it takes a finite amount of time to enter the interrupt service routine and execute it, the program adds 0x80 to TMR1H so that the next Timer1 Overflow interrupt occurs exactly in one second. If the program simply cleared TMR1L and set TMR1H to 0x80, the real-time clock would now be off by the finite amount of time previously described. Finally, the Timer1 Overflow interrupt flag is reset.

PORTB

The wake-up on change feature of PORTB was specifically designed to interface keys or a keypad directly to the microcontroller. Internal weak pull-up resistors are provided to reduce external parts count, while adding only a small amount of current. This feature allows the microcontroller to remain in the low-power SLEEP mode until a key is pressed. The device will wake-up from SLEEP when the key is pressed. The interrupt service routine will process the key input. The following example routine shows how to initialize the PORTB wake-up on change interrupt.

```
OPTION.RBPU = 0;
Temp = PORTB;
INTCON.RBIF = 0;
INTCON.RBIE = 1;
```

Whenever the state of the RB7:RB4 pins change, a mismatch condition occurs inside the microcontroller. The only way to clear the mismatch condition is to read PORTB, which also allows the RBIF interrupt flag to be cleared. In the previous code, the first line enables the internal weak pull-up resistors. The second line of the code resets the mismatch condition. Then the following lines clear the interrupt flag and enable the interrupt. A sample interrupt service routine is given below.

```
if(INTCON.RBIF)
{
   Temp = PORTB;
   Delay_Ms_4MHz(16);  //key debounce
   if(Temp!=0xf0 && Temp==PORTB)
   {
      StartBEEP(); //key press starts
beep

      if(!Temp.SET)        //SET key
         Flags.SET = 1;
      if(!Temp.UP)         //UP key
         Flags.UP = 1;
      if(!Temp.SOUND)      //SOUND key
      {
         if(Flags.SOUND_STATE)
            Flags.SOUND_STATE = 0;
         else
            Flags.SOUND_STATE = 1;
      }
   }
   INTCON.RBIF = 0;        //clear flag
}
```

The first line of the above example is used to verify that a change on PORTB interrupt has occurred. The next line is a 5 ms delay followed by a read of PORTB. This resets the mismatch condition. The following 20 ms delay is used in conjunction with the previous 5 ms delay for switch debouncing. The value of `Temp` and PORTB are compared and if equal a key press has been detected. The if statement also checks to see if the mismatch condition is from a key press or when the key is released. The key inputs are processed only when a key is pressed. When a key is pressed, it grounds the respective input pin. The nested if statements check each of the individual keys to see which have been pressed. If the SOUND key has been pressed, it merely toggles whether the hourly beep is enabled. Finally, the RBIF interrupt flag is cleared.

This application takes advantage of the wake-up on change and internal pull-up resistors to implement the SET, ▲, and SOUND keys. The SET key puts the clock in program mode and the PROG icon appears on the LCD. Program mode allows the user to set the time, AM or PM, and day of the week (see LCD Module for details on the LCD). Once in program mode, the hours digits flash. Pressing the ▲ key increments `Hours` from 12 to 11 AM and then from 12 to 11 PM. At any point the SET key can be pressed to advance to the minutes digits. The minutes digits are incremented using the ▲ key. `Minutes` can be incremented from 0 to 59. Pressing the SET key again flashes the day of the week. Using the ▲ key increments the day of the week from SU to SA. Finally, pressing the SET key takes the user out of program mode. If no key presses are detected for five seconds the program exits program mode.

CCP

The CCP module is used in Pulse Width Modulation (PWM) mode. The PWM signal is used to drive a piezo alarm circuit. The operation of PWM mode will not be discussed since it has been explained in great depth in application notes AN531, AN538, AN539, AN564, and AN594.

An example of configuring the CCP as a PWM output is shown below.

```
CCP1CON = 0x0f;       //Set CCP to PWM
PR2 = 122;            //period of 2048 Hz
CCPR1L = 5;           //Duty Cycle very low
TRISC.RC2 = 0;        //PWM pin output
T2CON = 0b01111101;   //Enable timer2
PIR1.TMR2IF = 0;      //Clear flag
PIE1.TMR2IE = 1;      //Enable interrupt
```

The CCP1CON register is set to 0x0F which puts the CCP into PWM mode. The period or frequency of the PWM output is set using the PR2 register. The following formula is used to calculate the value in PR2.

$$\text{PWM period} = [\ (\text{PR2}) + 1\ ] \cdot 4 \cdot \text{Tosc} \cdot (\text{TMR2 prescale value})\ ]$$

Another step in configuring the CCP is to set the value of the T2CON register. In the case above, a value of 0x7D configures the Timer2 output postscaler to 1:16, enables Timer2, and sets the Timer2 clock prescaler to 4. Using PR2 = 122, Tosc = 250 ns, and TMR2 prescale = 4, the resultant PWM period is approximately 500 µs or 2 kHz. This is the resonant frequency of the piezo alarm. The duty cycle, or more specifically the time the PWM output stays high, of the PWM output is set by the value of CCPR1L and bits CCP1CON<5:4>. The following formula uses the 10-bit value of CCPR1L:CCP1CON<5:4> to calculate the duty cycle.

$$\text{PWM duty cycle} = [\ (\text{CCPR1L:CCP1CON<5:4>}) \cdot \text{Tosc} \cdot (\text{TMR2 prescale value})\ ]$$

Using 0x014 as the value of CCPR1L:CCP1CON<5:4>, Tosc = 250 ns, and a TMR2 prescale = 4, the PWM duty cycle is calculated to be 20 µs or 4% duty cycle. The TRISC register must also be configured such that the PWM pin is setup as an output.

The Timer2 Overflow interrupt is used to turn off the PWM output. This produces a "beep" when the keys are pressed or an hourly alarm. The Timer2 postscaler waits for 16 PWM periods before the interrupt occurs. The following interrupt service routine is an example of how to generate a "beep".

```
if(PIR1.TMR2IF)
{
   Count--;
   if(!Count)
   {
      CCP1CON = 0;       //Disable CCP, and
      T2CON = 0;         //Timer2 and TMR2
      PIE1.TMR2IE = 0;   //interrupts
      CCPR1L = 0;
   }
   PIR1.TMR2IF = 0;      //Clear flag
}
```

The first line of code detects if a Timer2 Overflow interrupt has occurred. The variable `Count` is used to vary the length of the beep. `Count` is set previous to enabling the PWM. If `Count` reaches zero the following occurs:

- The PWM is disabled by clearing the CCP1CON register
- Timer2 is disabled by clearing the T2CON register
- The Timer2 Overflow interrupt is disabled by clearing the TMR2IE bit of the register PIE1
- The duty cycle register is cleared

Before exiting the service routine, the TMR2IF interrupt flag is cleared.

© 1997 Microchip Technology Inc.

A/D CONVERTER

Since the PIC16C924 has a five channel, 8-bit A/D converter and the LCD has a ° (degree) symbol and two digits, the application circuit has a thermistor for measuring temperature. Thermistors typically take hundreds of milliseconds to stabilize at a particular temperature and therefore the A/D converter is ideal for temperature measurements. Another feature of the A/D converter is the on-chip RC oscillator that can be used as the conversion clock. This feature allows the A/D to operate in SLEEP. The following code segment is used to initialize the A/D converter.

```
ADCON0 = 0b11000001;   //Enable A/D
ADCON1 = 0b00000100;   //Configure D/A I/O
PIR1.ADIF = 0;         //Clear flag
PIE1.ADIE = 1;         //Enable interrupt
```

The first line enables the internal RC for conversion clock, channel 0, and enables the A/D converter. The second line makes PORTA<1:0> analog inputs and PORTA<5,3:2> as digital I/O. The following lines clear the A/D conversion complete interrupt flag and enable the interrupt. The application uses the Timer1 Overflow interrupt to start a conversion every second using the GO bit of the ADCON0 register. The A/D conversion complete interrupt then processes the result of the conversion. The following is an example of the service routine.

```
if(PIR1.ADIF)
{
   TempC = ThermTable[ADRES];
   PIR1.ADIF = 0;
}
```

The first line checks for the ADIF interrupt flag. A lookup table, `ThermTable`, is used to convert the A/D result into a temperature reading. This table was created using calibration data from the thermistor. Finally, the ADIF interrupt flag is cleared.

The following code was added to the Timer1 interrupt service routine to start the A/D conversion.

```
TRISA.THERM_GND = 0;
DELAY_10_µs_4MHz(4);
ADCON0.GO = 1
NOP();
NOP();
TRISA.THERM_GND = 1;
```

The first line of code grounds the I/O pin connected to resistor R1. The 40 µs delay is provided so the A/D converter can sample the input signal. The A/D converter is then instructed to convert by setting the GO bit. A two cycle delay is added so that the sampling capacitor is disconnected from the input. Finally the I/O pin connected to R1 is made an input. This scheme only powers the thermistor when sampling, to reduce power consumption.

© 1997 Microchip Technology Inc.

LCD MODULE

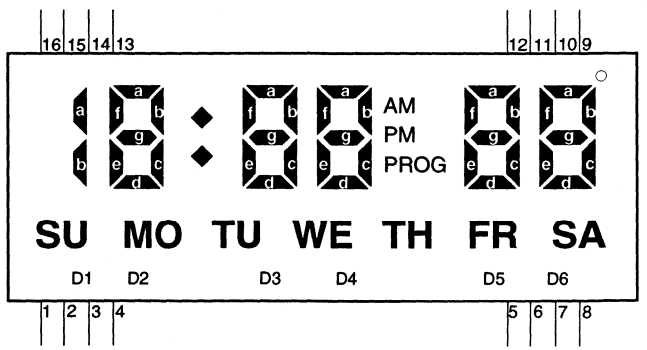
The LCD Module has a wealth of features typically found on more expensive dedicated LCD driver devices. The following is a detailed list of features:

- LCD Timing Sources
  - Fosc/256 (internal system clock)
  - Timer1 oscillator
  - Internal RC oscillator
- LCD Voltage Generation
  - Internal charge pump
  - External resistor ladder
- Bias
  - Static
  - 1/3
- Multiplex
  - Static
  - 1/2
  - 1/3
  - 1/4
- Operation during SLEEP
  - Only with internal RC or Timer1 clock sources
- Capable of up to 4 commons and 32 segments
  - 1 COM x 32 SEGs, total of 32 pixels
  - 2 COMs x 31 SEGs, total of 62 pixels
  - 3 COMs x 30 SEGs, total of 90 pixels
  - 4 COMs x 29 SEGs, total of 116 pixels
- 16 x 8 LCD data registers

The LCD Module is ideal for systems that use one controller board that has several applications with different displays. Microchip's OTP technology means that the controller board can be assembled with blank microcontrollers, and at final test the device can be programmed depending on the display type. This helps to reduce overhead and creates one board to track in inventory.

The particular LCD panel that was selected for this application is shown in Figure 2. It has four common electrodes and 12 segment electrodes. The panel provides 3 1/2 digits, colon, and AM/PM icons for time display. The panel also has day of the week icons (SU,MO,TU,WE,TH,FR,SA). This was the basis of another clock design using the PIC16C924.

FIGURE 4: LCD PINOUT



	COM0, pin 1	COM1, pin 8	COM2, pin 9	COM3, pin 16
SEG0, pin 2	SU	D2.e	D2.f	D1
SEG1, pin 3	MO	D3.e	:	D2.b
SEG2, pin 4	TU	D3.c	D3.b	D4.f
SEG3, pin 5	WE	D4.c	D4.b	AM
SEG4, pin 6	FR	D5.c	D6.f	D5.b
SEG5, pin 7	SA	D6.c	D6.b	°
SEG6, pin 10	D6.d	D6.e	D6.g	D6.a
SEG7, pin 11	D5.d	D5.e	D5.g	D5.a
SEG8, pin 12	TH	PROG	PM	D5.f
SEG9, pin 13	D4.d	D4.e	D4.g	D4.a
SEG10, pin 14	D3.d	D3.g	D3.f	D3.a
SEG11, pin 15	D2.d	D2.c	D2.g	D2.a

© 1997 Microchip Technology Inc.

The LCD Module is initialized by the following code:

```
STATUS.RP1 = 1;          //Change to Bank 2
LCDPS = 6;               //Frame freq to 37Hz
LCDSE = 0xff;            //All LCD I/O as LCD
LCDCON = 0b00010111;     //1/4 MUX charge pump
LCDD00 = 0;              //Clear all LCD
LCDD01 = 0;              //data RAM
LCDD02 = 0;
LCDD03 = 0;
LCDD04 = 0;
LCDD05 = 0;
LCDD06 = 0;
LCDD07 = 0;
LCDD08 = 0;
LCDD09 = 0;
LCDD10 = 0;
LCDD11 = 0;
LCDD12 = 0;
LCDD13 = 0;
LCDD14 = 0;
LCDD15 = 0;
LCDCON.LCDEN = 1;        //Enable LCD module
STATUS.RP1 = 0;          //Back to Bank 0
PIR1.LCDIF = 0;          //Clear flag
PIE1.LCDIE = 1;          //Enable interrupt
```

Note: At the time when this application note was generated, four banks of data memory were not supported by the MPC or MPLAB-C compilers.

The PIC16C924 starts a new page in Microchip history with four banks of data memory for the PIC16CXXX mid-range products. The first line of code switches to the second set of banks. The second line of code sets a frame frequency of approximately 37 Hz using the Timer1 oscillator. This frequency can be calculated by using the following formula:

$$\text{Clock source} / (128 \bullet (\text{LP3:LP0} + 1))$$

Using 32.768 kHz as the clock source and LP3:LP0 = 6 the resultant frame frequency is 36.57 Hz. Setting LCDSE to 0xFF configures ports D,E,F, and G as LCD drivers. Setting LCDCON to 0x17 configures the LCD Module for 1/4 MUX, 1/3 Bias, Timer1 clock source, charge pump enabled, and the LCD module will continue to drive during SLEEP. The LCD data registers LCDD00 - LCDD15 are all cleared, which turns off all pixels on the LCD panel. Any unused bits in the LCD data registers can be used as general purpose RAM. The PIC16C924 Clock can use the upper 4-bits of registers LCDD01, LCDD05, LCDD09, LCDD13 and all of LCDD02, LCDD03, LCDD06, LCDD07, LCDD10, LCDD11, LCDD14, LCDD15. This is a result of not using segments 13 through 28. The LCD Module is then enabled by setting the LCDEN bit in the LCDCON register. Finally the LCD interrupt flag is cleared and the interrupt is enabled.

The following is an example of the LCD interrupt service routine.

```
if(PIR1.LCDIF)
{
   Flags.FRAME = 1;
   PIR1.LCDIF = 0;
}
```

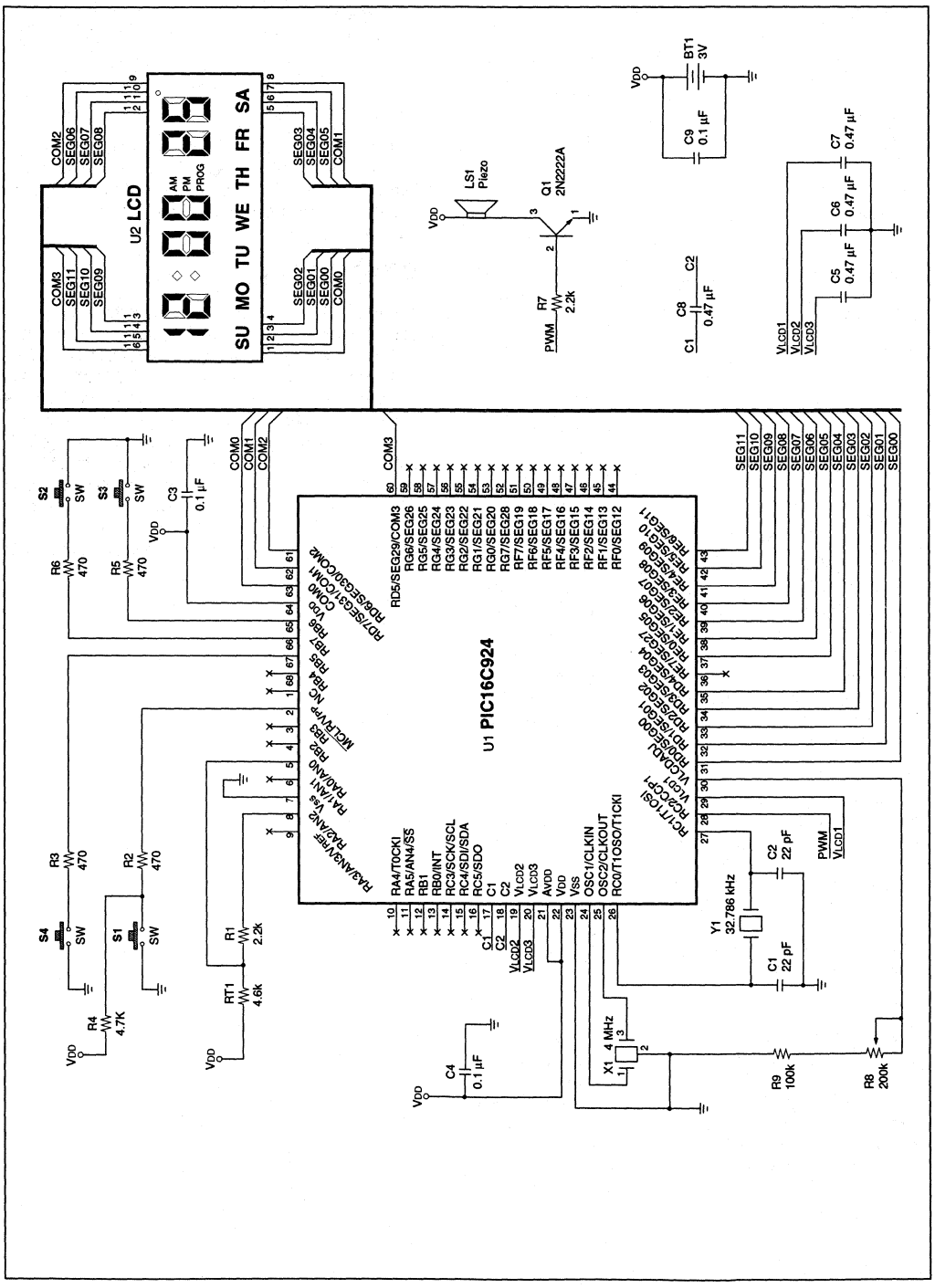
In this application, the LCD interrupt is used to signal the main routine that the LCD data registers can be updated without causing any flicker on the panel. Finally, as in any interrupt service routine, the interrupt flag is cleared.

CONCLUSION

The PIC16C924 is ideally suited towards battery applications such as thermostats or meters. It can extend battery life while maintaining a rich feature set such as A/D converter, SPI/I^2C module, and CCP module. All of the peripherals that have been previously described can operate during SLEEP, thus lowering the average current consumption. If the 8-bit A/D converter is not sufficient or not used in your design, Microchip also offers the PIC16C923 which has all the features of the PIC16C924 without the A/D converter. This device can use an external higher resolution A/D converter that can be interfaced to the SSP module, specifically the SPI port.

© 1997 Microchip Technology Inc.

APPENDIX A: PIC16C924 CLOCK SCHEMATIC



© 1997 Microchip Technology Inc.

APPENDIX B: PIC16C924 CLOCK FIRMWARE LISTING

```
                /*****************************************************************************
                 *  Filename: CLK.C
                 *****************************************************************************
                 *   Author:      Rodger Richey
                 *   Company:  Microchip Technology Incorporated
                 *   Revision: A3
                 *   Date:     12-17-96
                 *   Compiled using MPLAB-C Version 00.00.14
                 *****************************************************************************
                 *   Include files:
                 *   16C924.h    Rev 1.00
                 *   MUSIC.C     Rev A2
                 *   LCD.C       Rev A1
                 *   TIME.C      Rev A1
                 *****************************************************************************
                 *   Peripheral Modules Used:
                 *     Timer0    : Used by the music generation program for timing notes
                 *                 and durations
                 *     Timer1    : Used as a real time clock using an external 32.768KHz
                 *                 crystal and (2) 33pF capacitors
                 *     Timer2    : Used in conjunction with the PWM
                 *     CCP       : Used in PWM mode, for driving the piezo alarm
                 *     PORTB     : Used to decode key presses
                 *     A/D       : Used to measure temperature via a thermistor on RA0
                 *     LCD       : Used to display time, temperature, day of week
                 *****************************************************************************
                 *   External Clock Frequency    : 4MHz
                 *   Timer1 OSC Frequency        : 32.768KHz
                 *   Configuration Bit Settings  : XT Oscillator
                 *                               : Watchdog Timer OFF
                 *                               : Code Protect OFF
                 *                               : Power-Up Timer ON
                 *   Program Memory Usage        : 1095 words
                 *   Data Memory Usage           : 19 bytes
                 *****************************************************************************
                 *   Revision History
                 *   A1 - First Release
                 *   A2 - Added code to clear CCPR1L after PWM has finished
                 *      - Set No sleep flag in StartMusic
                 *   A3 - Changed __INT interrupt service routine.
                 *   Added INTCON.T0IE to check for Timer0 interrupt
                 *****************************************************************************
                 *   Note: Make sure that the temporary variables in the 16c924.h file
                 *         have been changed to locations 0x7a to 0x7f
                 *****************************************************************************/
                #include <16c924.h>
                #include <delay14.h>

                // PORTA pin defines
0002            #define THERM_GND 2

                // PORTB key defines
0004            #define EXTRA 4
0005            #define SOUND 5
0006            #define UP 6
0007            #define SET 7

                // Variable declarations
0026            bits Flags;                 // Contains flag bits for various events
0027            bits Temp;                  // Temporary storage
0028            bits Count;                 // Number of Timer2 Interrupts to count
0029            bits Ticks;                 // Counts seconds will in program mode
0078            bits Mode @ 0x78;           // Contains flag bits to turn on LCD pixels
002A            unsigned char FrameCnt;     // Count LCD frames
```

```
                // Bit defines for Flags
0000            #define UPDATE 0
0001            #define FRAME 1
0003            #define PROGRAM 3
0004            #define SOUND_STATE 4
0005            #define SLEEP_STATE 5

                // Time variables
0070            unsigned char Seconds @ 0x70;   // Holds number of seconds
0071            unsigned char Minutes @ 0x71;   // Holds number of minutes
0072            unsigned char Hours @ 0x72;     // Holds number of hours
0073            bits LStatus @ 0x73;            // bit 7 -> 0=AM, 1=PM
                                                // bits 0-3 -> 0000=SUN
0074            bits DayOfWeek @0x74;           //  0001=MON
                                                //  0010=TUE
                                                //  0011=WED
                                                //  0100=THU
                                                //  0101=FRI
                                                //  0110=SAT

                // Bit defines for LStatus
0007            #define AMPM 7

                // Temperature Variable
0075            unsigned char TempC @ 0x75;

000C            #define FRAME_COUNT 12          // Number of frames in blink in prgm mode
0004            #define BEEP_COUNT 4            // PWM duty cycle value for "beep"

                // Thermistor calibration table, 0C to 99C
                const unsigned int ThermTable[] =
            {
                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
                0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x02,0x03,0x04,
                0x05,0x06,0x06,0x07,0x08,0x09,0x09,0x10,0x11,0x11,0x12,0x12,0x13,0x13,0x14,0x14,
                0x15,0x15,0x16,0x16,0x17,0x17,0x18,0x18,0x19,0x19,0x20,0x20,0x21,0x21,0x22,0x22,
                0x23,0x23,0x24,0x24,0x25,0x25,0x25,0x26,0x26,0x27,0x27,0x28,0x28,0x28,0x29,0x29,
                0x30,0x30,0x30,0x31,0x31,0x32,0x32,0x32,0x33,0x33,0x33,0x34,0x34,0x35,0x35,0x35,
                0x36,0x36,0x37,0x37,0x38,0x38,0x38,0x39,0x39,0x40,0x40,0x40,0x41,0x41,0x42,0x42,
                0x43,0x43,0x43,0x44,0x44,0x45,0x45,0x45,0x46,0x46,0x47,0x47,0x48,0x48,0x48,0x49,
                0x49,0x50,0x50,0x50,0x51,0x51,0x52,0x52,0x53,0x53,0x54,0x54,0x55,0x55,0x56,0x56,
                0x57,0x57,0x58,0x58,0x59,0x59,0x60,0x60,0x61,0x61,0x62,0x62,0x63,0x63,0x64,0x64,
                0x65,0x65,0x66,0x66,0x67,0x67,0x68,0x68,0x69,0x69,0x70,0x71,0x71,0x72,0x73,0x73,
                0x74,0x74,0x75,0x76,0x76,0x77,0x78,0x78,0x79,0x79,0x80,0x81,0x81,0x82,0x83,0x84,
                0x84,0x85,0x86,0x87,0x88,0x89,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,
                0x99,0x99,0x99,0x99,0x99,0x99,0x99,0x99,0x99,0x99,0x99,0x99,0x99,0x99,0x99,0x99,
                0x99,0x99,0x99,0x99,0x99,0x99,0x99,0x99,0x99,0x99
            };
0005 0782     ADDWF   02
0006 3400     RETLW   00h
0007 3400     RETLW   00h
0008 3400     RETLW   00h
0009 3400     RETLW   00h
000A 3400     RETLW   00h
000B 3400     RETLW   00h
000C 3400     RETLW   00h
000D 3400     RETLW   00h
000E 3400     RETLW   00h
000F 3400     RETLW   00h
0010 3400     RETLW   00h
0011 3400     RETLW   00h
0012 3400     RETLW   00h
0013 3400     RETLW   00h
0014 3400     RETLW   00h
```


© 1997 Microchip Technology Inc.

```
0015 3400    RETLW  00h
0016 3400    RETLW  00h
0017 3400    RETLW  00h
0018 3400    RETLW  00h
0019 3400    RETLW  00h
001A 3400    RETLW  00h
001B 3400    RETLW  00h
001C 3400    RETLW  00h
001D 3400    RETLW  00h
001E 3400    RETLW  00h
001F 3400    RETLW  00h
0020 3400    RETLW  00h
0021 3400    RETLW  00h
0022 3400    RETLW  00h
0023 3400    RETLW  00h
0024 3400    RETLW  00h
0025 3400    RETLW  00h
0026 3400    RETLW  00h
0027 3400    RETLW  00h
0028 3400    RETLW  00h
0029 3400    RETLW  00h
002A 3400    RETLW  00h
002B 3400    RETLW  00h
002C 3400    RETLW  00h
002D 3400    RETLW  00h
002E 3400    RETLW  00h
002F 3400    RETLW  00h
0030 3400    RETLW  00h
0031 3400    RETLW  00h
0032 3401    RETLW  01h
0033 3402    RETLW  02h
0034 3403    RETLW  03h
0035 3404    RETLW  04h
0036 3405    RETLW  05h
0037 3406    RETLW  06h
0038 3406    RETLW  06h
0039 3407    RETLW  07h
003A 3408    RETLW  08h
003B 3409    RETLW  09h
003C 3409    RETLW  09h
003D 3410    RETLW  10h
003E 3411    RETLW  11h
003F 3411    RETLW  11h
0040 3412    RETLW  12h
0041 3412    RETLW  12h
0042 3413    RETLW  13h
0043 3413    RETLW  13h
0044 3414    RETLW  14h
0045 3414    RETLW  14h
0046 3415    RETLW  15h
0047 3415    RETLW  15h
0048 3416    RETLW  16h
0049 3416    RETLW  16h
004A 3417    RETLW  17h
004B 3417    RETLW  17h
004C 3418    RETLW  18h
004D 3418    RETLW  18h
004E 3419    RETLW  19h
004F 3419    RETLW  19h
0050 3420    RETLW  20h
0051 3420    RETLW  20h
0052 3421    RETLW  21h
0053 3421    RETLW  21h
0054 3422    RETLW  22h
0055 3422    RETLW  22h
0056 3423    RETLW  23h
```


© 1997 Microchip Technology Inc.

```
0057 3423    RETLW  23h
0058 3424    RETLW  24h
0059 3424    RETLW  24h
005A 3425    RETLW  25h
005B 3425    RETLW  25h
005C 3425    RETLW  25h
005D 3426    RETLW  26h
005E 3426    RETLW  26h
005F 3427    RETLW  27h
0060 3427    RETLW  27h
0061 3428    RETLW  28h
0062 3428    RETLW  28h
0063 3428    RETLW  28h
0064 3429    RETLW  29h
0065 3429    RETLW  29h
0066 3430    RETLW  30h
0067 3430    RETLW  30h
0068 3430    RETLW  30h
0069 3431    RETLW  31h
006A 3431    RETLW  31h
006B 3432    RETLW  32h
006C 3432    RETLW  32h
006D 3432    RETLW  32h
006E 3433    RETLW  33h
006F 3433    RETLW  33h
0070 3433    RETLW  33h
0071 3434    RETLW  34h
0072 3434    RETLW  34h
0073 3435    RETLW  35h
0074 3435    RETLW  35h
0075 3435    RETLW  35h
0076 3436    RETLW  36h
0077 3436    RETLW  36h
0078 3437    RETLW  37h
0079 3437    RETLW  37h
007A 3438    RETLW  38h
007B 3438    RETLW  38h
007C 3438    RETLW  38h
007D 3439    RETLW  39h
007E 3439    RETLW  39h
007F 3440    RETLW  40h
0080 3440    RETLW  40h
0081 3440    RETLW  40h
0082 3441    RETLW  41h
0083 3441    RETLW  41h
0084 3442    RETLW  42h
0085 3442    RETLW  42h
0086 3443    RETLW  43h
0087 3443    RETLW  43h
0088 3443    RETLW  43h
0089 3444    RETLW  44h
008A 3444    RETLW  44h
008B 3445    RETLW  45h
008C 3445    RETLW  45h
008D 3445    RETLW  45h
008E 3446    RETLW  46h
008F 3446    RETLW  46h
0090 3447    RETLW  47h
0091 3447    RETLW  47h
0092 3448    RETLW  48h
0093 3448    RETLW  48h
0094 3448    RETLW  48h
0095 3449    RETLW  49h
0096 3449    RETLW  49h
0097 3450    RETLW  50h
0098 3450    RETLW  50h
```


© 1997 Microchip Technology Inc.

```
0099 3450    RETLW  50h
009A 3451    RETLW  51h
009B 3451    RETLW  51h
009C 3452    RETLW  52h
009D 3452    RETLW  52h
009E 3453    RETLW  53h
009F 3453    RETLW  53h
00A0 3454    RETLW  54h
00A1 3454    RETLW  54h
00A2 3455    RETLW  55h
00A3 3455    RETLW  55h
00A4 3456    RETLW  56h
00A5 3456    RETLW  56h
00A6 3457    RETLW  57h
00A7 3457    RETLW  57h
00A8 3458    RETLW  58h
00A9 3458    RETLW  58h
00AA 3459    RETLW  59h
00AB 3459    RETLW  59h
00AC 3460    RETLW  60h
00AD 3460    RETLW  60h
00AE 3461    RETLW  61h
00AF 3461    RETLW  61h
00B0 3462    RETLW  62h
00B1 3462    RETLW  62h
00B2 3463    RETLW  63h
00B3 3463    RETLW  63h
00B4 3464    RETLW  64h
00B5 3464    RETLW  64h
00B6 3465    RETLW  65h
00B7 3465    RETLW  65h
00B8 3466    RETLW  66h
00B9 3466    RETLW  66h
00BA 3467    RETLW  67h
00BB 3467    RETLW  67h
00BC 3468    RETLW  68h
00BD 3468    RETLW  68h
00BE 3469    RETLW  69h
00BF 3469    RETLW  69h
00C0 3470    RETLW  70h
00C1 3471    RETLW  71h
00C2 3471    RETLW  71h
00C3 3472    RETLW  72h
00C4 3473    RETLW  73h
00C5 3473    RETLW  73h
00C6 3474    RETLW  74h
00C7 3474    RETLW  74h
00C8 3475    RETLW  75h
00C9 3476    RETLW  76h
00CA 3476    RETLW  76h
00CB 3477    RETLW  77h
00CC 3478    RETLW  78h
00CD 3478    RETLW  78h
00CE 3479    RETLW  79h
00CF 3479    RETLW  79h
00D0 3480    RETLW  80h
00D1 3481    RETLW  81h
00D2 3481    RETLW  81h
00D3 3482    RETLW  82h
00D4 3483    RETLW  83h
00D5 3484    RETLW  84h
00D6 3484    RETLW  84h
00D7 3485    RETLW  85h
00D8 3486    RETLW  86h
00D9 3487    RETLW  87h
00DA 3488    RETLW  88h
```


© 1997 Microchip Technology Inc.

```
00DB 3489    RETLW  89h
00DC 3490    RETLW  90h
00DD 3491    RETLW  91h
00DE 3492    RETLW  92h
00DF 3493    RETLW  93h
00E0 3494    RETLW  94h
00E1 3495    RETLW  95h
00E2 3496    RETLW  96h
00E3 3497    RETLW  97h
00E4 3498    RETLW  98h
00E5 3499    RETLW  99h
00E6 3499    RETLW  99h
00E7 3499    RETLW  99h
00E8 3499    RETLW  99h
00E9 3499    RETLW  99h
00EA 3499    RETLW  99h
00EB 3499    RETLW  99h
00EC 3499    RETLW  99h
00ED 3499    RETLW  99h
00EE 3499    RETLW  99h
00EF 3499    RETLW  99h
00F0 3499    RETLW  99h
00F1 3499    RETLW  99h
00F2 3499    RETLW  99h
00F3 3499    RETLW  99h
00F4 3499    RETLW  99h
00F5 3499    RETLW  99h
00F6 3499    RETLW  99h
00F7 3499    RETLW  99h
00F8 3499    RETLW  99h
00F9 3499    RETLW  99h
00FA 3499    RETLW  99h
00FB 3499    RETLW  99h
00FC 3499    RETLW  99h
00FD 3499    RETLW  99h
00FE 3499    RETLW  99h
00FF 3499    RETLW  99h

                /*************************************************************************
                 *   StartBEEP
                 *   Function: This routine configures the necessary hardware to emit a
                 *             beep from the piezo
                 *************************************************************************/
                              void StartBEEP(void)
                              {
0100 1283    BCF    03,5              Flags.SLEEP_STATE = 1; // Don't let the 924 go to sleep
0101 16A6    BSF    26,5
0102 3004    MOVLW  04h               Count = BEEP_COUNT;    // Set Count for length of beep
0103 00A8    MOVWF  28
0104 300F    MOVLW  0Fh               CCP1CON = 0x0f;        // Set the CCP module to PWM
0105 0097    MOVWF  17
0106 307A    MOVLW  7Ah               PR2 = 122;             // Set the period to 2048Hz
0107 1683    BSF    03,5
0108 0092    MOVWF  12
0109 3003    MOVLW  03h               CCPR1L = 3;            // Set the duty cycle very low
010A 1283    BCF    03,5
010B 0095    MOVWF  15
010C 307D    MOVLW  7Dh               T2CON = 0b01111101;    // Enable Timer2
010D 0092    MOVWF  12
010E 108C    BCF    0C,1              PIR1.TMR2IF = 0;       // Clear the Timer2 Interrupt flag
010F 1683    BSF    03,5              PIE1.TMR2IE = 1;       // Enable the Timer2 Interrupt
0110 148C    BSF    0C,1
0111 0008    RETURN                   return;
                              }

                              // Include source files
```


© 1997 Microchip Technology Inc.

```
                    #include "lcd.c"          // Contains programs that control the LCD
                    /*****************************************************************************
                     *  Filename: LCD.C
                     *****************************************************************************
                     *   Author:     Rodger Richey
                     *   Company:  Microchip Technology Incorporated
                     *   Revision: A0
                     *   Date:     6-14-96
                     *   Compiled using MPLAB-C Version 00.00.14
                     *****************************************************************************
                     *   This file contains routines to output time, day of week, AM/PM, and
                     *   temperature.  It also contains routines to blink the different groupings
                     *   of numbers, i.e. hours,seconds,day of the week.  Finally, the last
                     *   routine displays the state of the hourly beep.
                     *****************************************************************************/

                     // Bit defines for the Mode variable, tells the UpdateLCD routine which
                     // groups of numbers to display
0000                 #define HOURS 0
0001                 #define MINUTES 1
0002                 #define DAYOFWEEK 2
0005                 #define PROG 5           // PROG icon
0006                 #define COLON 6          // colon, :, for the hours/seconds time display
0007                 #define DEGREES 7        // degrees symbol for temperature

                    /*****************************************************************************
                     *   UpdateLCD
                     *   Function: This function updates the LCD display based on the Mode
                     *          variable.
                     *****************************************************************************/
                     void UpdateLCD(void)
                     {
                        // Array of 7-segment numbers
                                                          //   gfedcba
                        const unsigned char SevenSegTable[16]={ 0b00111111, // Zero
                                                                0b00000110, // One
                                                                0b01011011, // Two
                                                                0b01001111, // Three
                                                                0b01100110, // Four
                                                                0b01101101, // Five
                                                                0b01111101, // Six
                                                                0b00000111, // Seven
                                                                0b01111111, // Eight
                                                                0b01101111, // Nine
                                                                0b01110111, // Ten
                                                                0b01111100, // Eleven
                                                                0b01011000, // Twelve
                                                                0b01011110, // Thirteen
                                                                0b01111001, // Fourteen
0112 2924    GOTO   0124h                                       0b01110001};// Fifteen
0113 0782    ADDWF  02
0114 343F    RETLW  3Fh
0115 3406    RETLW  06h
0116 345B    RETLW  5Bh
0117 344F    RETLW  4Fh
0118 3466    RETLW  66h
0119 346D    RETLW  6Dh
011A 347D    RETLW  7Dh
011B 3407    RETLW  07h
011C 347F    RETLW  7Fh
011D 346F    RETLW  6Fh
011E 3477    RETLW  77h
011F 347C    RETLW  7Ch
0120 3458    RETLW  58h
0121 345E    RETLW  5Eh
0122 3479    RETLW  79h
```

```
0123 3471    RETLW  71h

                                    // Temporary variables in common RAM locations
0076                                 bits segment @ 0x76;
0077                                 unsigned char index @ 0x77;

                                    // Change to Bank 2, and clear all LCD data RAM
0124 1703    BSF    03,6             STATUS.RP1 = 1;
0125 1283    BCF    03,5             LCDD00 = 0;
0126 0190    CLRF   10
0127 0191    CLRF   11               LCDD01 = 0;
0128 0194    CLRF   14               LCDD04 = 0;
0129 0195    CLRF   15               LCDD05 = 0;
012A 0198    CLRF   18               LCDD08 = 0;
012B 0199    CLRF   19               LCDD09 = 0;
012C 019C    CLRF   1C               LCDD12 = 0;
012D 019D    CLRF   1D               LCDD13 = 0;

                                    // Update day of the week if enabled
012E 1D78    BTFSS  78,2              if(Mode.DAYOFWEEK)
012F 2965    GOTO   0165h
0130                                 {
                                         // Update Day of the Week
0130 08F4    MOVF   74                   if(DayOfWeek == 0)              // Sunday
0131 1D03    BTFSS  03,2
0132 2936    GOTO   0136h
0133 1283    BCF    03,5                     LCDD00.0 = 1;
0134 1410    BSF    10,0
0135 2965    GOTO   0165h                else if(DayOfWeek == 1)         // Monday
0136 3001    MOVLW  01h                      LCDD00.1 = 1;
0137 1283    BCF    03,5
0138 0274    SUBWF  74,W
0139 1D03    BTFSS  03,2
013A 293E    GOTO   013Eh
013B 1283    BCF    03,5
013C 1490    BSF    10,1
013D 2965    GOTO   0165h                else if(DayOfWeek == 2)         // Tuesday
013E 3002    MOVLW  02h                      LCDD00.2 = 1;
013F 1283    BCF    03,5
0140 0274    SUBWF  74,W
0141 1D03    BTFSS  03,2
0142 2946    GOTO   0146h
0143 1283    BCF    03,5
0144 1510    BSF    10,2
0145 2965    GOTO   0165h                else if(DayOfWeek == 3)         // Wednesday
0146 3003    MOVLW  03h                      LCDD00.3 = 1;
0147 1283    BCF    03,5
0148 0274    SUBWF  74,W
0149 1D03    BTFSS  03,2
014A 294E    GOTO   014Eh
014B 1283    BCF    03,5
014C 1590    BSF    10,3
014D 2965    GOTO   0165h                else if(DayOfWeek == 4)         // Thursday
014E 3004    MOVLW  04h                      LCDD01.0 = 1;
014F 1283    BCF    03,5
0150 0274    SUBWF  74,W
0151 1D03    BTFSS  03,2
0152 2956    GOTO   0156h
0153 1283    BCF    03,5
0154 1411    BSF    11,0
0155 2965    GOTO   0165h                else if(DayOfWeek == 5)         // Friday
0156 3005    MOVLW  05h                      LCDD00.4 = 1;
0157 1283    BCF    03,5
0158 0274    SUBWF  74,W
0159 1D03    BTFSS  03,2
015A 295E    GOTO   015Eh
```


© 1997 Microchip Technology Inc.

```
015B 1283    BCF    03,5
015C 1610    BSF    10,4
015D 2965    GOTO   0165h                  else if(DayOfWeek == 6)        // Saturday
015E 3006    MOVLW  06h                        LCDD00.5 = 1;
015F 1283    BCF    03,5
0160 0274    SUBWF  74,W
0161 1D03    BTFSS  03,2
0162 2965    GOTO   0165h
0163 1283    BCF    03,5
0164 1690    BSF    10,5
                                        }

                                        // Update Time if enabled
0165 1283    BCF    03,5                if(Mode.HOURS)
0166 1C78    BTFSS  78,0
0167 2988    GOTO   0188h
0168                                    {
                                            // Update AM/PM icons
0168 1FF3    BTFSS  73,7                 if(LStatus.AMPM)
0169 296C    GOTO   016Ch
016A 1419    BSF    19,0                         LCDD09.0 = 1;
016B 296D    GOTO   016Dh                else
016C 159C    BSF    1C,3                         LCDD12.3 = 1;

                                            // Digit 1
016D 1A72    BTFSC  72,4                 if(Hours&0x10)
016E 141C    BSF    1C,0                     LCDD12.0 = 1;

                                            // Digit 2
016F 300F    MOVLW  0Fh                  index = Hours & 0x0f;
0170 0572    ANDWF  72,W
0171 00F7    MOVWF  77
0172 00FB    MOVWF  7B                   segment = SevenSegTable[index];
0173 3001    MOVLW  01h
0174 008A    MOVWF  0A
0175 08FB    MOVF   7B
0176 0877    MOVF   77,W
0177 2113    CALL   0113h
0178 1283    BCF    03,5
0179 00F6    MOVWF  76
017A 1876    BTFSC  76,0                 if(segment.0)                  // D2.a
017B 159D    BSF    1D,3                     LCDD13.3 = 1;
017C 18F6    BTFSC  76,1                 if(segment.1)                  // D2.b
017D 149C    BSF    1C,1                     LCDD12.1 = 1;
017E 1976    BTFSC  76,2                 if(segment.2)                  // D2.c
017F 1595    BSF    15,3                     LCDD05.3 = 1;
0180 19F6    BTFSC  76,3                 if(segment.3)                  // D2.d
0181 1591    BSF    11,3                     LCDD01.3 = 1;
0182 1A76    BTFSC  76,4                 if(segment.4)                  // D2.e
0183 1414    BSF    14,0                     LCDD04.0 = 1;
0184 1AF6    BTFSC  76,5                 if(segment.5)                  // D2.f
0185 1418    BSF    18,0                     LCDD08.0 = 1;
0186 1B76    BTFSC  76,6                 if(segment.6)                  // D2.g
0187 1599    BSF    19,3                     LCDD09.3 = 1;
                                        }

                                        // Update Minutes if enabled
0188 1CF8    BTFSS  78,1                 if(Mode.MINUTES)
0189 29C6    GOTO   01C6h
018A                                    {
                                            // Digit 3
018A 30F0    MOVLW  F0h                  index = Minutes & 0xf0;
018B 0571    ANDWF  71,W
018C 00F7    MOVWF  77
018D 1003    BCF    03,0                     index >>= 4;
018E 0CF7    RRF    77
```

```
018F 1003    BCF    03,0
0190 0CF7    RRF    77
0191 1003    BCF    03,0
0192 0CF7    RRF    77
0193 1003    BCF    03,0
0194 0CF7    RRF    77
0195 00FB    MOVWF  7B                    segment = SevenSegTable[index];
0196 3001    MOVLW  01h
0197 008A    MOVWF  0A
0198 08FB    MOVF   7B
0199 0877    MOVF   77,W
019A 2113    CALL   0113h
019B 1283    BCF    03,5
019C 00F6    MOVWF  76
019D 1876    BTFSC  76,0                  if(segment.0)                    // D3.a
019E 151D    BSF    1D,2                      LCDD13.2 = 1;
019F 18F6    BTFSC  76,1                  if(segment.1)                    // D3.b
01A0 1518    BSF    18,2                      LCDD08.2 = 1;
01A1 1976    BTFSC  76,2                  if(segment.2)                    // D3.c
01A2 1514    BSF    14,2                      LCDD04.2 = 1;
01A3 19F6    BTFSC  76,3                  if(segment.3)                    // D3.d
01A4 1511    BSF    11,2                      LCDD01.2 = 1;
01A5 1A76    BTFSC  76,4                  if(segment.4)                    // D3.e
01A6 1494    BSF    14,1                      LCDD04.1 = 1;
01A7 1AF6    BTFSC  76,5                  if(segment.5)                    // D3.f
01A8 1519    BSF    19,2                      LCDD09.2 = 1;
01A9 1B76    BTFSC  76,6                  if(segment.6)                    // D3.g
01AA 1515    BSF    15,2                      LCDD05.2 = 1;

                                          // Digit 4
01AB 300F    MOVLW  0Fh                   index = Minutes & 0x0f;
01AC 0571    ANDWF  71,W
01AD 00F7    MOVWF  77
01AE 00FB    MOVWF  7B                    segment = SevenSegTable[index];
01AF 3001    MOVLW  01h
01B0 008A    MOVWF  0A
01B1 08FB    MOVF   7B
01B2 0877    MOVF   77,W
01B3 118A    BCF    0A,3
01B4 2113    CALL   0113h
01B5 118A    BCF    0A,3
01B6 1283    BCF    03,5
01B7 00F6    MOVWF  76
01B8 1876    BTFSC  76,0                  if(segment.0)                    // D4.a
01B9 149D    BSF    1D,1                      LCDD13.1 = 1;
01BA 18F6    BTFSC  76,1                  if(segment.1)                    // D4.b
01BB 1598    BSF    18,3                      LCDD08.3 = 1;
01BC 1976    BTFSC  76,2                  if(segment.2)                    // D4.c
01BD 1594    BSF    14,3                      LCDD04.3 = 1;
01BE 19F6    BTFSC  76,3                  if(segment.3)                    // D4.d
01BF 1491    BSF    11,1                      LCDD01.1 = 1;
01C0 1A76    BTFSC  76,4                  if(segment.4)                    // D4.e
01C1 1495    BSF    15,1                      LCDD05.1 = 1;
01C2 1AF6    BTFSC  76,5                  if(segment.5)                    // D4.f
01C3 151C    BSF    1C,2                      LCDD12.2 = 1;
01C4 1B76    BTFSC  76,6                  if(segment.6)                    // D4.g
01C5 1499    BSF    19,1                      LCDD09.1 = 1;
                                         }

                                         // Update Temperature
                                         // Digit 5
01C6 30F0    MOVLW  F0h                   index = TempC&0xf0;
01C7 0575    ANDWF  75,W
01C8 00F7    MOVWF  77
01C9 1003    BCF    03,0                  index >>= 4;
01CA 0CF7    RRF    77
```


© 1997 Microchip Technology Inc.

```
01CB 1003    BCF    03,0
01CC 0CF7    RRF    77
01CD 1003    BCF    03,0
01CE 0CF7    RRF    77
01CF 1003    BCF    03,0
01D0 0CF7    RRF    77
01D1 00FB    MOVWF  7B          segment = SevenSegTable[index];
01D2 3001    MOVLW  01h
01D3 008A    MOVWF  0A
01D4 08FB    MOVF   7B
01D5 0877    MOVF   77,W
01D6 2113    CALL   0113h
01D7 1283    BCF    03,5
01D8 00F6    MOVWF  76
01D9 1876    BTFSC  76,0        if(segment.0)                    // D5.a
01DA 179C    BSF    1C,7                LCDD12.7 = 1;
01DB 18F6    BTFSC  76,1        if(segment.1)                    // D5.b
01DC 161C    BSF    1C,4                LCDD12.4 = 1;
01DD 1976    BTFSC  76,2        if(segment.2)                    // D5.c
01DE 1614    BSF    14,4                LCDD04.4 = 1;
01DF 19F6    BTFSC  76,3        if(segment.3)                    // D5.d
01E0 1790    BSF    10,7                LCDD00.7 = 1;
01E1 1A76    BTFSC  76,4        if(segment.4)                    // D5.e
01E2 1794    BSF    14,7                LCDD04.7 = 1;
01E3 1AF6    BTFSC  76,5        if(segment.5)                    // D5.f
01E4 141D    BSF    1D,0                LCDD13.0 = 1;
01E5 1B76    BTFSC  76,6        if(segment.6)                    // D5.g
01E6 1798    BSF    18,7                LCDD08.7 = 1;

                                // Digit 6
01E7 300F    MOVLW  0Fh         index = TempC&0x0f;
01E8 0575    ANDWF  75,W
01E9 00F7    MOVWF  77
01EA 00FB    MOVWF  7B          segment = SevenSegTable[index];
01EB 3001    MOVLW  01h
01EC 008A    MOVWF  0A
01ED 08FB    MOVF   7B
01EE 0877    MOVF   77,W
01EF 118A    BCF    0A,3
01F0 2113    CALL   0113h
01F1 118A    BCF    0A,3
01F2 1283    BCF    03,5
01F3 00F6    MOVWF  76
01F4 1876    BTFSC  76,0        if(segment.0)                    // D6.a
01F5 171C    BSF    1C,6                LCDD12.6 = 1;
01F6 18F6    BTFSC  76,1        if(segment.1)                    // D6.b
01F7 1698    BSF    18,5                LCDD08.5 = 1;
01F8 1976    BTFSC  76,2        if(segment.2)                    // D6.c
01F9 1694    BSF    14,5                LCDD04.5 = 1;
01FA 19F6    BTFSC  76,3        if(segment.3)                    // D6.d
01FB 1710    BSF    10,6                LCDD00.6 = 1;
01FC 1A76    BTFSC  76,4        if(segment.4)                    // D6.e
01FD 1714    BSF    14,6                LCDD04.6 = 1;
01FE 1AF6    BTFSC  76,5        if(segment.5)                    // D6.f
01FF 1618    BSF    18,4                LCDD08.4 = 1;
0200 1B76    BTFSC  76,6        if(segment.6)                    // D6.g
0201 1718    BSF    18,6                LCDD08.6 = 1;

                                // Turn on : if enabled
0202 1B78    BTFSC  78,6        if(Mode.COLON)
0203 1498    BSF    18,1                LCDD08.1 = 1;

                                // Turn on degrees symbol if enabled
0204 1BF8    BTFSC  78,7        if(Mode.DEGREES)
0205 169C    BSF    1C,5                LCDD12.5 = 1;
```


© 1997 Microchip Technology Inc.

```
                                        // Turn on PROG symbol if enabled
0206 1AF8   BTFSC  78,5                 if(Mode.PROG)
0207 1415   BSF    15,0                         LCDD05.0 = 1;

                                        // Make copies of the LCD data registers
0208 0810   MOVF   10,W                 LCDD02 = LCDD00;
0209 0092   MOVWF  12
020A 0811   MOVF   11,W                 LCDD03 = LCDD01;
020B 0093   MOVWF  13
020C 0814   MOVF   14,W                 LCDD06 = LCDD04;
020D 0096   MOVWF  16
020E 0815   MOVF   15,W                 LCDD07 = LCDD05;
020F 0097   MOVWF  17
0210 0818   MOVF   18,W                 LCDD10 = LCDD08;
0211 009A   MOVWF  1A
0212 0819   MOVF   19,W                 LCDD11 = LCDD09;
0213 009B   MOVWF  1B
0214 081C   MOVF   1C,W                 LCDD14 = LCDD12;
0215 009E   MOVWF  1E
0216 081D   MOVF   1D,W                 LCDD15 = LCDD13;
0217 009F   MOVWF  1F

0218 1303   BCF    03,6                 STATUS.RP1 = 0;          // Return to Bank 0
0219 0008   RETURN                      return;
                                    }

                /*************************************************************************
                 *   BlinkLCD
                 *   Function: This function is used in program mode to blink hours, minutes
                 *             or day of the week depending on the current status.
                 *************************************************************************/
                                    void BlinkLCD(bits which)
002B                                {
021A 1283   BCF    03,5
021B 00AB   MOVWF  2B
021C 1C26   BTFSS  26,0                 if(Flags.UPDATE)     // If UPDATE flag is set, blank the
021D 2A3C   GOTO   023Ch
021E                                    {                            // corresponding group
021E 1026   BCF    26,0                     Flags.UPDATE = 0;        // Clear UPDATE flag
021F 3001   MOVLW  01h                      if(which==0x01)          // Blank Hours
0220 022B   SUBWF  2B,W
0221 1D03   BTFSS  03,2
0222 2A28   GOTO   0228h
0223                                        {
0223 30E6   MOVLW  E6h                          Mode = 0b11100110;
0224 1283   BCF    03,5
0225 00F8   MOVWF  78
0226 2112   CALL   0112h                       UpdateLCD();
                                            }
0227 2A3B   GOTO   023Bh                    else if(which == 0x02)   // Blank Minutes
0228 3002   MOVLW  02h                     {
0229 1283   BCF    03,5
022A 022B   SUBWF  2B,W
022B 1D03   BTFSS  03,2
022C 2A32   GOTO   0232h
022D 30E5   MOVLW  E5h                          Mode = 0b11100101;
022E 1283   BCF    03,5
022F 00F8   MOVWF  78
0230 2112   CALL   0112h                       UpdateLCD();
                                            }
0231 2A3B   GOTO   023Bh                    else if(which == 0x04)   // Blank day of the week
0232 3004   MOVLW  04h                     {
0233 1283   BCF    03,5
0234 022B   SUBWF  2B,W
0235 1D03   BTFSS  03,2
0236 2A3B   GOTO   023Bh
```


© 1997 Microchip Technology Inc.

```
0237 30E3    MOVLW  E3h                      Mode = 0b11100011;
0238 1283    BCF    03,5
0239 00F8    MOVWF  78
023A 2112    CALL   0112h                    UpdateLCD();
                                         }
                                     }
023B 2A40    GOTO   0240h                else                    // Turn on all groups if UPDATE
flag is
                                     {                           // cleared
023C 1426    BSF    26,0                 Flags.UPDATE = 1;       // Set the UPDATE flag
023D 30E7    MOVLW  E7h                  Mode = 0b11100111;      // Set bits in Mode to turn all on
023E 00F8    MOVWF  78
023F 2112    CALL   0112h                UpdateLCD();
                                     }
0240 0008    RETURN                      return;
                                 }

                 /*************************************************************************
                  *   DisplaySoundState
                  *   Function: When the SOUND button is pressed the state of the hourly beep
                  *           is toggle between on and off.  This routine displays to the
                  *            *          user the state, Snd OF or Snd On.
                  *************************************************************************/
                                 void DisplaySoundState(void)
                                 {
002C                                 unsigned char frames;   // Frame count variable

0241 300C    MOVLW  0Ch                  frames = FRAME_COUNT;   // Initialize the frame counter
0242 1283    BCF    03,5
0243 00AC    MOVWF  2C

0244 1283    BCF    03,5                 while(!Flags.FRAME);    // Wait for next frame to occur
0245 1CA6    BTFSS  26,1
0246 2A44    GOTO   0244h
0247 1283    BCF    03,5                 Flags.FRAME = 0;        // Clear FRAME flag
0248 10A6    BCF    26,1

0249 1703    BSF    03,6                 STATUS.RP1 = 1;         // Change to Bank 2
024A 3080    MOVLW  80h                  LCDD00 = 0b10000000;    // Write data common to both
024B 0090    MOVWF  10
024C 300A    MOVLW  0Ah                  LCDD01 = 0b00001010;    // Snd On and Snd OF
024D 0091    MOVWF  11
024E 300E    MOVLW  0Eh                  LCDD05 = 0b00001110;
024F 0095    MOVWF  15
0250 300A    MOVLW  0Ah                  LCDD09 = 0b00001010;
0251 0099    MOVWF  19
0252 3009    MOVLW  09h                  LCDD13 = 0b00001001;
0253 009D    MOVWF  1D
0254 1303    BCF    03,6                 STATUS.RP1 = 0;         // Change to Bank 0

0255 1E26    BTFSS  26,4                 if(Flags.SOUND_STATE)   // Sound on
0256 2A5F    GOTO   025Fh
0257                                     {
0257 1703    BSF    03,6                     STATUS.RP1 = 1;     // Change to Bank 2
0258 30FE    MOVLW  FEh                      LCDD04 = 0b11111110; // Write data specific to Snd On
0259 0094    MOVWF  14
025A 3049    MOVLW  49h                      LCDD08 = 0b01001001;
025B 0098    MOVWF  18
025C 3090    MOVLW  90h                      LCDD12 = 0b10010000;
025D 009C    MOVWF  1C
                                     }
025E 2A66    GOTO   0266h                else                    // Sound off
                                     {
025F 1703    BSF    03,6                     STATUS.RP1 = 1;     // Change to Bank 2
0260 30DE    MOVLW  DEh                      LCDD04 = 0b11011110;// Write data specific to Snd OFF
0261 0094    MOVWF  14
```

```
0262 3059    MOVLW  59h                  LCDD08 = 0b01011001;
0263 0098    MOVWF  18
0264 30D0    MOVLW  D0h                  LCDD12 = 0b11010000;
0265 009C    MOVWF  1C
                                      }
0266 1303    BCF    03,6                STATUS.RP1 = 0;          // Change to Bank 0

                                      while(frames)    // Delay a specific number of frames so
                                      {                        // the user can see the message
0267 1283    BCF    03,5
0268 08AC    MOVF   2C
0269 1903    BTFSC  03,2
026A 2A71    GOTO   0271h
026B                                      if(Flags.FRAME)      // Decrement frames until 0
026B 1283    BCF    03,5                  {
026C 1CA6    BTFSS  26,1
026D 2A70    GOTO   0270h
026E 10A6    BCF    26,1                      Flags.FRAME = 0;
026F 03AC    DECF   2C                        frames--;
                                          }
0270 2A67    GOTO   0267h                }
0271 0008    RETURN                      return;
                                   }

                                   #include "time.c"       // Contains programs for timekeeping
                          /*************************************************************************
                           *  Filename: TIME.C
                           *************************************************************************
                           *   Author:     Rodger Richey
                           *   Company:  Microchip Technology Incorporated
                           *   Revision: A0
                           *   Date:     6-14-96
                           *   Compiled using MPLAB-C Version 00.00.14
                           *************************************************************************
                           *   This file contains the routines to increment time and set the time in
                           *   program mode.
                           *************************************************************************/

                          /*************************************************************************
                           *   IncMinutes
                           *   Function: Increments minutes and performs checks.  Minutes is in BCD.
                           *************************************************************************/
                                   void IncMinutes(void)
                                   {
0272 1283    BCF    03,5                Seconds = 0;             // Clear Seconds
0273 01F0    CLRF   70
0274 0AF1    INCF   71                  Minutes++;               // Increment Minutes
0275 300F    MOVLW  0Fh                 if( (Minutes&0x0f) > 0x09) // Check for BCD overflow
0276 0571    ANDWF  71,W
0277 00FB    MOVWF  7B
0278 3009    MOVLW  09h
0279 027B    SUBWF  7B,W
027A 1D03    BTFSS  03,2
027B 1C03    BTFSS  03,0
027C 2A82    GOTO   0282h
027D                                   {
027D 30F0    MOVLW  F0h                    Minutes &= 0xf0;
027E 1283    BCF    03,5
027F 05F1    ANDWF  71
0280 3010    MOVLW  10h                    Minutes += 0x10;
0281 07F1    ADDWF  71
                                       }
0282 0008    RETURN                    return;
                                   }

                          /*************************************************************************
```


© 1997 Microchip Technology Inc.

```
                *   IncHours
                *   Function: Increments hours and performs checks.  Hours is in BCD.
                ********************************************************************************/
                                    void IncHours(void)
                                    {
0283 1283       BCF     03,5            if(!Flags.PROGRAM) // If program mode, do not clear Minutes
0284 1DA6       BTFSS   26,3
0285 01F1       CLRF    71                 Minutes = 0;
0286 0AF2       INCF    72              Hours++;                    // Increment Hours
0287 300F       MOVLW   0Fh             if( (Hours&0x0f) > 0x09)    // Check for BCD overflow
0288 0572       ANDWF   72,W
0289 00FB       MOVWF   7B
028A 3009       MOVLW   09h
028B 027B       SUBWF   7B,W
028C 1D03       BTFSS   03,2
028D 1C03       BTFSS   03,0
028E 2A94       GOTO    0294h
028F                                {
028F 30F0       MOVLW   F0h             Hours &= 0xf0;
0290 1283       BCF     03,5
0291 05F2       ANDWF   72
0292 3010       MOVLW   10h             Hours += 0x10;
0293 07F2       ADDWF   72
                                    }
0294 3012       MOVLW   12h         if(Hours == 0x12)      // If hours = 12 then change AM/PM
0295 1283       BCF     03,5
0296 0272       SUBWF   72,W
0297 1D03       BTFSS   03,2
0298 2AA8       GOTO    02A8h
0299                                {                       //  and day of the week accordingly
0299 1283       BCF     03,5        if(LStatus.AMPM)    // If PM
029A 1FF3       BTFSS   73,7
029B 2AA7       GOTO    02A7h
029C                                {
029C 13F3       BCF     73,7          LStatus.AMPM = 0;  // change to AM
029D 19A6       BTFSC   26,3          if(!Flags.PROGRAM)  // If not prog mode, increment day
029E 2AA6       GOTO    02A6h
029F                                  {               // of the week and check for overflow
029F 0AF4       INCF    74             DayOfWeek++;
02A0 3007       MOVLW   07h            if(DayOfWeek == 0x07)
02A1 0274       SUBWF   74,W
02A2 1D03       BTFSS   03,2
02A3 2AA6       GOTO    02A6h
02A4 1283       BCF     03,5                DayOfWeek = 0;
02A5 01F4       CLRF    74
                                      }
                                    }
02A6 2AA8       GOTO    02A8h       else                // Otherwise change to PM
02A7 17F3       BSF     73,7          LStatus.AMPM = 1;
                                    }
02A8 1283       BCF     03,5        if(Flags.SOUND_STATE) // Start hourly beep if enabled
02A9 1E26       BTFSS   26,4
02AA 2AAC       GOTO    02ACh
02AB 2100       CALL    0100h           StartBEEP();

02AC 3013       MOVLW   13h         if(Hours == 0x13)    // If hours has overflowed, set to 1
02AD 1283       BCF     03,5
02AE 0272       SUBWF   72,W
02AF 1D03       BTFSS   03,2
02B0 2AB4       GOTO    02B4h
02B1 3001       MOVLW   01h             Hours = 0x01;
02B2 1283       BCF     03,5
02B3 00F2       MOVWF   72

02B4 0008       RETURN              return;
                                    }
```

```
                    /*************************************************************************
                     *   SetTime
                     *   Function: This routine sets the time in program mode.  Allows hours,
                     *          minutes and day of the week to be configured.
                     *************************************************************************/
                                  void SetTime(void)
                                  {
02B5 300C    MOVLW  0Ch               FrameCnt = FRAME_COUNT; // Initialize the frame counter
02B6 1283    BCF    03,5
02B7 00AA    MOVWF  2A
                                      while(!Flags.SET)    // Wait for the SET button to be
02B8 1283    BCF    03,5              {                    // hit before advancing to minutes
02B9 1BA6    BTFSC  26,7
02BA 2AD2    GOTO   02D2h
02BB                                    if(Flags.UP)      // If UP button pressed, inc Minutes
02BB 1F26    BTFSS  26,6                {
02BC 2ABF    GOTO   02BFh
02BD 1326    BCF    26,6                 Flags.UP = 0;
02BE 2283    CALL   0283h                IncHours();
                                        }
02BF 1283    BCF    03,5                if(Flags.FRAME)  // Toggle display state (blink) every
02C0 1CA6    BTFSS  26,1
02C1 2ACC    GOTO   02CCh
02C2                                    {                  // FRAME_COUNT frames
02C2 10A6    BCF    26,1                 Flags.FRAME = 0;
02C3 03AA    DECF   2A                   FrameCnt--;
02C4 08AA    MOVF   2A                   if(!FrameCnt)  // If frame count = zero, toggle state
02C5 1D03    BTFSS  03,2
02C6 2ACC    GOTO   02CCh
02C7                                      {
02C7 300C    MOVLW  0Ch                    FrameCnt = FRAME_COUNT;
02C8 1283    BCF    03,5
02C9 00AA    MOVWF  2A
02CA 3001    MOVLW  01h                    BlinkLCD(0x01);
02CB 221A    CALL   021Ah
                                        }
                                      }
02CC 3005    MOVLW  05h               if(Ticks == 5)        // If no button pressed in 5 secs
02CD 1283    BCF    03,5
02CE 0229    SUBWF  29,W
02CF 1903    BTFSC  03,2
02D0 0008    RETURN                       return;              // exit program mode
02D1 2AB8    GOTO   02B8h             }
02D2 1283    BCF    03,5              Flags.SET = 0;
02D3 13A6    BCF    26,7

02D4 300C    MOVLW  0Ch               FrameCnt = FRAME_COUNT; // Initialize the frame counter
02D5 00AA    MOVWF  2A
                                      while(!Flags.SET)    // Wait for the SET button to be hit
02D6 1283    BCF    03,5              {                  // before advancing to day of the week
02D7 1BA6    BTFSC  26,7
02D8 2AF7    GOTO   02F7h
02D9                                    if(Flags.UP)  // If the UP button is hit, inc Minutes
02D9 1F26    BTFSS  26,6                {
02DA 2AE4    GOTO   02E4h
02DB 1326    BCF    26,6                 Flags.UP = 0;
02DC 2272    CALL   0272h                IncMinutes();
02DD 3060    MOVLW  60h                  if(Minutes >= 0x60)     // Check for upper limit
02DE 1283    BCF    03,5
02DF 0271    SUBWF  71,W
02E0 1C03    BTFSS  03,0
02E1 2AE4    GOTO   02E4h
02E2 1283    BCF    03,5                 Minutes = 0;
02E3 01F1    CLRF   71
                                        }
```


© 1997 Microchip Technology Inc.

```
02E4 1283    BCF    03,5                if(Flags.FRAME)     // Toggle the display state (blink)
02E5 1CA6    BTFSS  26,1
02E6 2AF1    GOTO   02F1h
02E7                                     {                        // every FRAME_COUNT frames
02E7 10A6    BCF    26,1                  Flags.FRAME = 0;
02E8 03AA    DECF   2A                    FrameCnt--;
02E9 08AA    MOVF   2A                    if(!FrameCnt)     // If FRAME_COUNT=zero, toggle state
02EA 1D03    BTFSS  03,2
02EB 2AF1    GOTO   02F1h
02EC                                       {
02EC 300C    MOVLW  0Ch                     FrameCnt = FRAME_COUNT;
02ED 1283    BCF    03,5
02EE 00AA    MOVWF  2A
02EF 3002    MOVLW  02h                     BlinkLCD(0x02);
02F0 221A    CALL   021Ah
                                           }
                                         }
02F1 3005    MOVLW  05h                  if(Ticks == 5)      // If no button pressed in 5 secs
02F2 1283    BCF    03,5
02F3 0229    SUBWF  29,W
02F4 1903    BTFSC  03,2
02F5 0008    RETURN                        return;               // exit program mode
02F6 2AD6    GOTO   02D6h              }
02F7 1283    BCF    03,5               Flags.SET = 0;
02F8 13A6    BCF    26,7

02F9 300C    MOVLW  0Ch               FrameCnt = FRAME_COUNT;  // Initialize the frame counter
02FA 00AA    MOVWF  2A
                                      while(!Flags.SET)         // Wait for SET button to be hit
02FB 1283    BCF    03,5            {                          // before exiting program mode
02FC 1BA6    BTFSC  26,7
02FD 2B1B    GOTO   031Bh
02FE                                    if(Flags.UP)           // If the UP key is pressed, inc
02FE 1F26    BTFSS  26,6                {                       // inc. day of week
02FF 2B08    GOTO   0308h
0300 1326    BCF    26,6                 Flags.UP = 0;
0301 0AF4    INCF   74                   DayOfWeek++;
0302 3007    MOVLW  07h                  if(DayOfWeek == 0x07) // Check for overflow
0303 0274    SUBWF  74,W
0304 1D03    BTFSS  03,2
0305 2B08    GOTO   0308h                  DayOfWeek = 0;
0306 1283    BCF    03,5
0307 01F4    CLRF   74                  }
0308 1283    BCF    03,5                if(Flags.FRAME)        // Toggle display state every
0309 1CA6    BTFSS  26,1
030A 2B15    GOTO   0315h
030B                                    {                       // every FRAME_COUNT frames
030B 10A6    BCF    26,1                 Flags.FRAME = 0;
030C 03AA    DECF   2A                   FrameCnt--;
030D 08AA    MOVF   2A                   if(!FrameCnt)
030E 1D03    BTFSS  03,2
030F 2B15    GOTO   0315h                  {
0310                                        FrameCnt = FRAME_COUNT;
0310 300C    MOVLW  0Ch
0311 1283    BCF    03,5                   BlinkLCD(0x04);
0312 00AA    MOVWF  2A
0313 3004    MOVLW  04h                    }
0314 221A    CALL   021Ah                 }
                                         if(Ticks == 5)          // If no button pressed in 5 secs
0315 3005    MOVLW  05h
0316 1283    BCF    03,5
0317 0229    SUBWF  29,W                   return;           // exit program mode
0318 1903    BTFSC  03,2
0319 0008    RETURN
```

```
031A 2AFB    GOTO   02FBh            }
031B 1283    BCF    03,5             Flags.SET = 0;
031C 13A6    BCF    26,7
031D 0008    RETURN                  return;
                                   }

                /*************************************************************************
                 *   Init924
                 *   Function: This routine initializes the peripherals and CPU of the
                 *          PIC16C924.
                 *************************************************************************/
                           void Init924(void)
                           {
031E 1283    BCF    03,5    STATUS.RP0 = 0;          // Change to Bank 0
031F 1303    BCF    03,6    STATUS.RP1 = 0;
0320 3053    MOVLW  53h     OPTION = 0b01010011;     // Pull-ups on,T0 int clk source,
0321 1683    BSF    03,5
0322 0081    MOVWF  01
                                                     // Prescaler assigned to T0, 1:16
0323 30C1    MOVLW  C1h     ADCON0 = 0b11000001;     // Internal RC clk src, Ch0, A/D on
0324 1283    BCF    03,5
0325 009F    MOVWF  1F
0326 0185    CLRF   05      PORTA = 0;               // Clear ports A,B,C
0327 0186    CLRF   06      PORTB = 0;
0328 0187    CLRF   07      PORTC = 0;
0329 3007    MOVLW  07h     TRISA = 0b00000111;      // RAZ:RA1> digi outputs
032A 1683    BSF    03,5
032B 0085    MOVWF  05
032C 30F0    MOVLW  F0h     TRISB = 0xf0;            // Upper 4 pins are inputs for keys
032D 0086    MOVWF  06
032E 3003    MOVLW  03h     TRISC = 0x03;            // RC<0:1> used for Timer1
032F 0087    MOVWF  07                               // external crystal
0330 3004    MOVLW  04h     ADCON1 = 0b00000100;     // RA<0:1,3> are analog
0331 009F    MOVWF  1F
0332 1283    BCF    03,5    PIR1.ADIF = 0;           // Clear A/D interrupt flag
0333 130C    BCF    0C,6
0334 1683    BSF    03,5    PIE1.ADIE = 1;           // Enable A/D interrupt
0335 170C    BSF    0C,6

0336 1283    BCF    03,5    Temp = PORTB;            // Clear mismatch condition
0337 0806    MOVF   06,W                             // on PORTB
0338 00A7    MOVWF  27
0339 100B    BCF    0B,0    INTCON.RBIF = 0;         // Clear PORTB interrupt flag
033A 158B    BSF    0B,3    INTCON.RBIE = 1;         // Enable PORTB interrupt

033B 3080    MOVLW  80h     TMR1H = 0x80;            // Initialize Timer1 to 0x8000
033C 008F    MOVWF  0F
033D 018E    CLRF   0E      TMR1L = 0x00;
033E 300F    MOVLW  0Fh     T1CON = 0b00001111;      // Timer1 1:1 prescale, Osc
033F 0090    MOVWF  10                               // enabled, no sync, external
                                                     // clock source, Timer1 on
0340 100C    BCF    0C,0    PIR1.TMR1IF = 0;         // Clear Timer1 Overflow int flag
0341 1683    BSF    03,5    PIE1.TMR1IE = 1;         // Enable T1 Overflow interrupt
0342 140C    BSF    0C,0

0343 1703    BSF    03,6    STATUS.RP1 = 1;          // Go to Bank 2
0344 3006    MOVLW  06h     LCDPS = 6;               // Set LCD frame freq to 37 Hz,
0345 1283    BCF    03,5                            // Timer1 clk source
0346 008E    MOVWF  0E
0347 30FF    MOVLW  FFh     LCDSE = 0xff;            // Ports D,E,F,G are all LCD pins
0348 008D    MOVWF  0D
0349 3017    MOVLW  17h     LCDCON = 0b00010111;     // Drive in SLEEP,charge pump on,Timer1 clk src
034A 008F    MOVWF  0F                               // Timer1 clk src 1/4 mux, 1/3 bias
034B 0190    CLRF   10      LCDD00 = 0;              // Clear all LCD data registers
034C 0191    CLRF   11      LCDD01 = 0;
```


© 1997 Microchip Technology Inc.

```
034D 0192    CLRF   12              LCDD02 = 0;
034E 0193    CLRF   13              LCDD03 = 0;
034F 0194    CLRF   14              LCDD04 = 0;
0350 0195    CLRF   15              LCDD05 = 0;
0351 0196    CLRF   16              LCDD06 = 0;
0352 0197    CLRF   17              LCDD07 = 0;
0353 0198    CLRF   18              LCDD08 = 0;
0354 0199    CLRF   19              LCDD09 = 0;
0355 019A    CLRF   1A              LCDD10 = 0;
0356 019B    CLRF   1B              LCDD11 = 0;
0357 019C    CLRF   1C              LCDD12 = 0;
0358 019D    CLRF   1D              LCDD13 = 0;
0359 019E    CLRF   1E              LCDD14 = 0;
035A 019F    CLRF   1F              LCDD15 = 0;
035B 178F    BSF    0F,7            LCDCON.LCDEN = 1;    // Enable the LCD Module
035C 1303    BCF    03,6            STATUS.RP1 = 0;      // Go to Bank 0
035D 138C    BCF    0C,7            PIR1.LCDIF = 0;      // Clear LCD interrupt flag
035E 1683    BSF    03,5            PIE1.LCDIE = 1;      // Enable LCD interrupt
035F 178C    BSF    0C,7

0360 1283    BCF    03,5            Seconds = 0;         // Initialize data variables
0361 01F0    CLRF   70
0362 01F1    CLRF   71              Minutes = 0;
0363 3012    MOVLW  12h             Hours = 0x12;        // Set time to 12:00AM Sunday
0364 00F2    MOVWF  72
0365 01F3    CLRF   73              LStatus = 0;
0366 01F4    CLRF   74              DayOfWeek = 0;
0367 3011    MOVLW  11h             Flags = 0b00010001;
0368 00A6    MOVWF  26
0369 01F5    CLRF   75              TempC = 0;
036A 01A9    CLRF   29              Ticks = 0;
036B 3004    MOVLW  04h             Count = BEEP_COUNT;
036C 00A8    MOVWF  28
036D 30C7    MOVLW  C7h             Mode = 0b11000111;   // Turn on :,degrees,hours,
036E 00F8    MOVWF  78                                   // minutes, day of week

036F 170B    BSF    0B,6            INTCON.PEIE = 1;     // Enable peripheral interrupts
0370 178B    BSF    0B,7            INTCON.GIE = 1;      // Enable global interrupts
0371 0008    RETURN                 return;
                                }

                    /*************************************************************************
                     *   main
                     *   Function: Controls the clock.  Calls routines to update the LCD panel,
                     *             play music, program mode, and display state of sound.
                     *************************************************************************/
                                    void main(void)
                                    {
0372 231E    CALL   031Eh               Init924();           // Initialize the PIC16C924

                                       while(1)
                                       {
0373 1283    BCF    03,5                 if(Flags.UPDATE&&Flags.FRAME) // Refresh the LCD
0374 1C26    BTFSS  26,0                                               // data registers
0375 2B7D    GOTO   037Dh
0376 1CA6    BTFSS  26,1
0377 2B7D    GOTO   037Dh
0378                                    {                    // based on new data
0378 1283    BCF    03,5                 Flags.UPDATE = 0;    // Clear the UPDATE flag
0379 1026    BCF    26,0
037A 10A6    BCF    26,1                 Flags.FRAME = 0;     // Clear the FRAME flag
037B 2112    CALL   0112h                UpdateLCD();         // Update LCD data regs
                                       }
037C 2B84    GOTO   0384h               else if(!Flags.UPDATE&&Flags.FRAME)  // Clear FRAME
037D 1283    BCF    03,5                 Flags.FRAME = 0;                  // flag if no UPDATE
037E 1826    BTFSC  26,0
```

```
037F 2B84    GOTO   0384h
0380 1CA6    BTFSS  26,1
0381 2B84    GOTO   0384h
0382 1283    BCF    03,5
0383 10A6    BCF    26,1

0384 1283    BCF    03,5                if(Flags.SET)          // Enter program mode
0385 1FA6    BTFSS  26,7
0386 2B9A    GOTO   039Ah
0387                                     {
0387 13A6    BCF    26,7                  Flags.SET = 0;       // Clear the SET, UP flags
0388 1326    BCF    26,6                  Flags.UP = 0;
0389 01A9    CLRF   29                    Ticks = 0;           // Clear the Ticks
038A 15A6    BSF    26,3                  Flags.PROGRAM = 1;   // Change to program mode
038B 1283    BCF    03,5                  while(!Flags.FRAME);  // Wait for next frame to occur
038C 1CA6    BTFSS  26,1
038D 2B8B    GOTO   038Bh
038E 1283    BCF    03,5                  Flags.FRAME = 0;     // Clear FRAME flag
038F 10A6    BCF    26,1
0390 30E7    MOVLW  E7h                   Mode = 0b11100111;   // Enable PROG icon on LCD
0391 00F8    MOVWF  78
0392 2112    CALL   0112h                 UpdateLCD();         // Refresh the LCD

0393 22B5    CALL   02B5h                 SetTime();           // Call program to set time

0394 1283    BCF    03,5                  Flags.PROGRAM = 0;   // Exit program mode
0395 11A6    BCF    26,3
0396 01F0    CLRF   70                    Seconds = 0;         // Clear seconds
0397 1426    BSF    26,0                  Flags.UPDATE = 1;    // Set UPDATE flag
0398 30C7    MOVLW  C7h                   Mode = 0b11000111;   // Reset display mode,
0399 00F8    MOVWF  78                                         // PROG icon off
                                         }

039A 1EF3    BTFSS  73,5                 if(LStatus.SOUND)      // Enable/disable hourly beep
039B 2B9E    GOTO   039Eh
039C                                     {
039C 12F3    BCF    73,5                  LStatus.SOUND = 0;   // Reset SOUND flag
039D 2241    CALL   0241h                 DisplaySoundState(); // Display state of hourly beep
                                         }

039E 1283    BCF    03,5                 if(!Flags.SLEEP_STATE) // If 924 can go to sleep,
039F 1EA6    BTFSS  26,5                                       // go ahead
03A0 0063    SLEEP                          SLEEP();
03A1 2B73    GOTO   0373h              }
03A2 0008    RETURN                 }

                  /************************************************************************
                   *   __INT
                   *   Function: Interrupt service routine for LCD, PORTB, Timer2, Timer1,
                   *          Timer0, and A/D
                   ************************************************************************/
0004 2BA3    GOTO   03A3h          void __INT(void)
03A3                               {
                                   #asm                          // "push" W and STATUS
03A3 00FA                movwf   temp_WREG
03A4 0E03                swapf   STATUS,W
03A5 1283                bcf     STATUS,RP0
03A6 1303                bcf     STATUS,RP1
03A7 00FE                movwf   temp_STATUS
03A8 0804                movf   FSR,W
03A9 00FF                movwf  temp_FSR
                                   #endasm

03AA 1283    BCF    03,5              if(PIR1.LCDIF)            // Ok to write to LCD data regs
03AB 1F8C    BTFSS  0C,7
03AC 2BAF    GOTO   03AFh
```


© 1997 Microchip Technology Inc.

```
03AD                                    {
03AD 14A6    BSF    26,1                   Flags.FRAME = 1;     // Set FRAME flag
03AE 138C    BCF    0C,7                   PIR1.LCDIF = 0;      // Clear LCD interrupt flag
                                        }

03AF 1C0B    BTFSS  0B,0                 if(INTCON.RBIF)         // Key press/release detected
03B0 2BD7    GOTO   03D7h
03B1                                    {
03B1 3005    MOVLW  05h                     Delay_Ms_4MHz(5);    // Debounce for 5msec
03B2 2432    CALL   0432h
03B3 1283    BCF    03,5                    Temp = PORTB;        // Read PORTB
03B4 0806    MOVF   06,W
03B5 00A7    MOVWF  27
03B6 3014    MOVLW  14h                     Delay_Ms_4MHz(20);   // Debounce for 20msec more
03B7 2432    CALL   0432h
03B8 30F0    MOVLW  F0h                     if(Temp!=0xf0 && Temp==PORTB) // If same state as
03B9 1283    BCF    03,5                                                 // previous read
03BA 0227    SUBWF  27,W
03BB 1903    BTFSC  03,2
03BC 2BD4    GOTO   03D4h
03BD 1283    BCF    03,5
03BE 0827    MOVF   27,W
03BF 0206    SUBWF  06,W
03C0 1D03    BTFSS  03,2
03C1 2BD4    GOTO   03D4h
03C2                                       {                    // and interrupt is not for release
03C2 2100    CALL   0100h                    StartBEEP();        // Beep when key is pressed

03C3 1283    BCF    03,5                    if(!Temp.SET)       // Set the SET flag
03C4 1FA7    BTFSS  27,7
03C5 17A6    BSF    26,7                     Flags.SET = 1;
03C6 1F27    BTFSS  27,6                    if(!Temp.UP)        // Set the UP flag
03C7 1726    BSF    26,6                     Flags.UP = 1;
03C8 118A    BCF    0A,3                    if(!Temp.SOUND&&!Flags.PROGRAM)
03C9 1AA7    BTFSC  27,5
03CA 2BD4    GOTO   03D4h
03CB 19A6    BTFSC  26,3
03CC 2BD4    GOTO   03D4h
03CD                                       {                    // Toggle the SOUND state
03CD 1283    BCF    03,5                    if(Flags.SOUND_STATE)
03CE 1E26    BTFSS  26,4
03CF 2BD2    GOTO   03D2h
03D0 1226    BCF    26,4                       Flags.SOUND_STATE = 0;
03D1 2BD3    GOTO   03D3h                    else
03D2 1626    BSF    26,4                       Flags.SOUND_STATE = 1;
03D3 16F3    BSF    73,5                    LStatus.SOUND = 1;  // Set the SOUND flag
                                           }
                                        }
03D4 1283    BCF    03,5                    Ticks = 0;                // Reset Ticks, because key
03D5 01A9    CLRF   29                                               // was pressed
03D6 100B    BCF    0B,0                 INTCON.RBIF = 0;          // Clear PORTB interrupt flag
                                        }

03D7 1C8C    BTFSS  0C,1                 if(PIR1.TMR2IF&&PIE1.TMR2IE)// T2 Overflow used for beep
03D8 2BEE    GOTO   03EEh
03D9 1683    BSF    03,5
03DA 1C8C    BTFSS  0C,1
03DB 2BEE    GOTO   03EEh
03DC                                    {
03DC 1683    BSF    03,5                   if(PIE1.TMR2IE)           // If Timer2 int is enabled
03DD 1C8C    BTFSS  0C,1
03DE 2BEC    GOTO   03ECh
03DF                                       {
03DF 1283    BCF    03,5                    Count--;                 // Decrement count
03E0 03A8    DECF   28
03E1 08A8    MOVF   28                      if(!Count)              // If count has reached zero
```

```
03E2 1D03    BTFSS  03,2
03E3 2BEC    GOTO   03ECh
03E4                                    {
03E4 1283    BCF    03,5                 CCP1CON = 0;          // Disable CCP module
03E5 0197    CLRF   17
03E6 0192    CLRF   12                   T2CON = 0;            // Disable Timer2
03E7 0195    CLRF   15                   CCPR1L = 0;           // Clear the Duty Cycle
03E8 1683    BSF    03,5                 PIE1.TMR2IE = 0;      // Disable Timer2 Interrupt
03E9 108C    BCF    0C,1
03EA 1283    BCF    03,5                 Flags.SLEEP_STATE = 0;  // Enable 924 to SLEEP
03EB 12A6    BCF    26,5
                                        }
                                     }
03EC 1283    BCF    03,5              PIR1.TMR2IF = 0;         // Clear Timer2 interrupt flag
03ED 108C    BCF    0C,1
                                  }

03EE 1283    BCF    03,5           if(PIR1.TMR1IF)             // T1 Overflow, once every sec
03EF 1C0C    BTFSS  0C,0
03F0 2C23    GOTO   0423h
03F1                                {
03F1 19A6    BTFSC  26,3              if(!Flags.PROGRAM)       // If not in program mode
03F2 2C16    GOTO   0416h
03F3                                  {
03F3 0AF0    INCF   70                  Seconds++;             // Increment seconds
03F4 300F    MOVLW  0Fh                 if( (Seconds&0x0f) > 0x09)    // check for seconds
03F5 0570    ANDWF  70,W                                      // overflow within
03F6 00FB    MOVWF  7B
03F7 3009    MOVLW  09h
03F8 027B    SUBWF  7B,W
03F9 1D03    BTFSS  03,2
03FA 1C03    BTFSS  03,0
03FB 2C01    GOTO   0401h
03FC                                   {                      // seconds
03FC 30F0    MOVLW  F0h                  Seconds &= 0xf0;
03FD 1283    BCF    03,5
03FE 05F0    ANDWF  70
03FF 3010    MOVLW  10h                  Seconds += 0x10;
0400 07F0    ADDWF  70
                                       }
0401 3060    MOVLW  60h                 if(Seconds >= 0x60)   // check for seconds overflow
0402 1283    BCF    03,5
0403 0270    SUBWF  70,W
0404 1C03    BTFSS  03,0
0405 2C07    GOTO   0407h
0406 2272    CALL   0272h                IncMinutes();         // increment minutes routine
0407 3060    MOVLW  60h                 if(Minutes >= 0x60)   // check for hours overflow
0408 1283    BCF    03,5
0409 0271    SUBWF  71,W
040A 1C03    BTFSS  03,0
040B 2C0D    GOTO   040Dh
040C 2283    CALL   0283h                IncHours();           // increment hours routine

040D 1283    BCF    03,5                TMR1H |= 0x80;        // Set Timer1 to 0x8000
040E 178F    BSF    0F,7                                     // + current time
040F 1426    BSF    26,0                Flags.UPDATE = 1;     // Set UPDATE flag

0410 1F78    BTFSS  78,6                if(Mode.COLON)        // Toggle whether the colon is
0411 2C14    GOTO   0414h
0412 1378    BCF    78,6                 Mode.COLON = 0;      // on or off every second
0413 2C15    GOTO   0415h               else
0414 1778    BSF    78,6                 Mode.COLON = 1;
                                       }
0415 2C17    GOTO   0417h               else                  // If in program mode
0416 0AA9    INCF   29                    Ticks++;            // inc Ticks, used for timeout
```


© 1997 Microchip Technology Inc.

```
0417 1683    BSF    03,5                TRISA.THERM_GND = 0; // Apply power to thermistor
0418 1105    BCF    05,2
0419 3002    MOVLW  02h                 Delay_10xUs_4MHz(2); // Allow 20us for sampling
041A 243C    CALL   043Ch
041B 1283    BCF    03,5                ADCON0.GO = 1;       // Start a temp A/D conversion
041C 151F    BSF    1F,2
041D 0000    NOP                        NOP();               // Wait for charging cap to
041E 0000    NOP                        NOP();               // disconnect from pin
041F 1683    BSF    03,5                TRISA.THERM_GND = 1; // Remove power from thermistor
0420 1505    BSF    05,2
0421 1283    BCF    03,5                PIR1.TMR1IF = 0;     // Clear Timer1 interrupt flag
0422 100C    BCF    0C,0
                                       }

0423 1F0C    BTFSS  0C,6                if(PIR1.ADIF)        // A/D conversion complete
0424 2C2B    GOTO   042Bh
0425                                   {
0425 018A    CLRF   0A                    TempC = ThermTable[ADRES];// Use converted value
0426 081E    MOVF   1E,W                                    // for table
0427 2005    CALL   0005h
0428 1283    BCF    03,5
0429 00F5    MOVWF  75
042A 130C    BCF    0C,6                 PIR1.ADIF = 0;     // lookup of temperature
                                       }

                                  #asm                      // "pop" W and STATUS
042B 087F                   movf   temp_FSR,W
042C 0084                   movwf  FSR
042D 0E7E                   swapf   temp_STATUS,W
042E 0083                   movwf   STATUS
042F 0EFA                   swapf   temp_WREG,F
0430 0E7A                   swapf   temp_WREG,W
                                  #endasm

0431 0009    RETFIE                    return;
                                  }

                /*************************************************************************/

                                  void Delay_Ms_4MHz(registerw delay)
                                  /*
                                     Clock Speed = 4MHz
                                     Inst. Clock = 1MHz
                                     Inst. dur.  = 1us  */
0000                              {
                                  #asm
0432 1283                       BCF     STATUS, RP0
0433 00FB                       MOVWF   __WImage
                          DLMS4M1
                                RADIX   DEC
0434 30F9                       MOVLW   249
0435 0084                       MOVWF   FSR
                          DLMS4M2
0436 0000                       NOP
0437 0B84                       DECFSZ  FSR
0438 2C36                       GOTO    DLMS4M2

0439 0BFB                       DECFSZ  __WImage
043A 2C34                       GOTO    DLMS4M1
                                  #endasm
043B 0008    RETURN               }

                /*************************************************************************/

                                  void Delay_10xUs_4MHz(registerw delay)
                                  /*
```

```
                                    Clock Freq. = 4MHz
                                    Inst. Clock = 1MHz
                                    Inst. dur.  = 1000ns  */
0000                                {
                                    #asm
043C 1283                          BCF     STATUS, RP0
043D 00FB                          MOVWF   __WImage
                           DL10XMS4M
                           DL10XMS4M__  REPT 7
                                    NOP
                           ENDM
043E 0000
043F 00 00
0440 00 00
0441 00 00
0442 00 00
0443 00 00
0444 00 00
0445 FB 0B                      DECFSZ  __WImage
0446 3E 2C                      GOTO    DL10XMS4M
                                  #endasm
0447 0008    RETURN                 }
                        /*****************************************************************************/

0000 3003    MOVLW  03h
0001 008A    MOVWF  0A
0002 2B72    GOTO   0372h

ROM USAGE MAP

    0000 to 0002    0004 to 0447
    Total ROM used 0447

Errors              :    0
Warnings            :    0
```


© 1997 Microchip Technology Inc.

PWM, a Software Solution for the PIC16CXXX

Author: Ole Röpcke
Consultant, Europe

INTRODUCTION

The low cost, high performance features of a PIC16CXXX microcontroller make it a suitable device for automatic control technology applications. Sometimes, an additional PWM output is needed. For some devices, such as the PIC16C71, the addition of a software PWM adds the missing element. It is possible to use Timer0 (which also provides the system clock) and its corresponding interrupt to generate a PWM output with a duty cycle that can vary from nearly 10% to 90%. However, some applications require a greater duty cycle range.

This application note provides a software solution for a more accurate and flexible PWM output, which is characterized by the following:

1. PWM frequency up to 19.6 kHz (20 MHz crystal).
2. Variable duty cycle from 0% to 100%.
3. 8-bit resolution.
4. PWM step size of 1 TCY (Instruction Cycle Time).
5. Simultaneous generation of a system time clock.

METHODS

Before the solution is revealed, we must first examine the various software methods used to generate variable length pulses.

In the following explanations, the unit of time will be the length of an Instruction Cycle (TCY). We will use TCY because one instruction (if the program counter is not changed) executes in one TCY and Timer0 (without prescaling) increments every TCY. This provides us with a simple method to control a potentially complex timing sequence.

Making use of the time needed to execute an instruction provides a very simple method of generating an pulse. Example 1 (Method A) shows an instruction sequence, which will generate a high pulse of 99 TCY on pin 3 of PORTA. The pulse length is controlled by the value of register LENGTH in steps of 3 TCY. This is the computing time needed by one program loop.

The drawbacks of this method are an excessive use of computing time and a poor PWM resolution.

EXAMPLE 1: USING TCY AS A TIME BASE FOR PULSE GENERATION (METHOD A)

```
        .
        .
        .
        PORTA     equ  04h
        LENGTH    equ  0ch
        COUNTER   equ  0dh
        .
        .
        .
        movlw     34              ; value for pulse length of 99 Tcy
        movwf     LENGTH          ;
        .
        .
        .
        movf      LENGTH,W        ; write value to the loop counter
        movwf     COUNTER         ;
        bsf       PORTA,3         ; start high pulse
Loop
        decfsz    COUNTER,F       ; counting pulse length
        goto      Loop            ;
        bcf       PORTA,3         ; end high pulse
        .
        .
        .
```

© 1997 Microchip Technology Inc.

However, the architectural features of Microchip's midrange microcontrollers allow us to proceed in another direction. Example 2 shows an instruction sequence (Method B), which enables us to generate a high pulse with lengths varying from 1 to 5 TCY. The addition of any number to the file register PCL increases the program counter and skips a predetermined number of instructions (depending on the number added to PCL). The length of the high pulse is the same as the computing time consumed by the number of executed BSF instructions and is controlled by the value of file register LENGTH. If LENGTH is set to 4, 4 BSF instructions will be skipped and a high pulse of 1 TCY will be generated. If LENGTH is set to 0, no instructions will be skipped and the length of the pulse remains 5 TCY. A special effect takes place, when LENGTH is set to 5. All BSF instructions are skipped and therefore no high pulse occurs. This instruction sequence is able to generate pulses of various lengths including the length 0 TCY. Between each length the step size is only 1 TCY. This method will generate a long pulse if LENGTH is set to a small value, and a short pulse if LENGTH is set to a large value.

The drawback of this method is that it is suitable only for short pulses. Long pulses require an excessive amount of ROM and computing time.

EXAMPLE 2: PULSE GENERATION OPTIMIZED TO DEVICE ARCHITECTURE (METHOD B)

```
.
.
.
PORTA     equ  04h
LENGTH    equ  0ch
.
.
.
movlw     4              ; value for pulse length of 1 Tcy
movwf     LENGTH         ;
.
.
.
movf      LENGTH,W       ; This is an indirectly addressed relative jump
addwf     PCL,F          ;
bsf       PORTA,3        ; start high pulse 5 Tcy
bsf       PORTA,3        ; start high pulse 4 Tcy
bsf       PORTA,3        ; start high pulse 3 Tcy
bsf       PORTA,3        ; start high pulse 2 Tcy
bsf       PORTA,3        ; start high pulse 1 Tcy
bcf       PORTA,3        ; end high pulse
.
.
.
```


© 1997 Microchip Technology Inc.

A third method (Method C) uses the Timer0 Interrupt. After the port is set, the timer is loaded with an adjusted number corresponding to the desired pulse length. If the timer overflows, the interrupt service routine starts and resets the port. The number, which has to be loaded, is defined by the following factors:

- the counting direction of the timer (up/down)
- the time between loading the timer and setting the port
- the timer clock cycles needed to write to the timer
- the computing time between timer overflow and executing the first instruction of the interrupt service routine
- the computing time of the interrupt service routine until the port is reset
- the desired pulse length
- an additional correction number, which is related to the method of calculation

Method C is able to generate very different pulse lengths without wasting any computing time. This method is specifically useful to generate long pulses. When the prescaler is not used, the available step size is 1 TCY. But this procedure is unsuitable for generating very short pulses, because every interrupt service routine needs a minimum of computing time for execution, which defines the minimum pulse length.

Every PWM signal is a continuous succession of high and low pulses. The length of each pulse is defined by the desired duty cycle. Adding the length of a low and a high pulse gives us the PWM frequency, which should be kept constant.

Method C will only work if each of the pulses are longer than the minimum computing time of the interrupt service routine. The interrupt service routine has to find the required pulse and set or reset the port. In addition to that the interrupt service routine has to calculate the number required by the timer and write that value to the timer.

Whether it is feasible to use the timer interrupt depends on the desired pulse length and the known minimum computing time of the interrupt service routine. If Method C will not give the desired PWM signal, one has to fall back on methods A or B as described in Example 1 and Example 2.

Each method has its advantages and disadvantages. Because all three methods are software based, it is possible to change methods at any time. Therefore the microcontroller can determine, even during the interrupt service routine, which method is best suited to generate the next pulse. This is the underlying concept for this application.

The software PWM module uses methods B and C, which are shown in Table 1. If possible, both pulses are generated by the timer interrupt and two timer interrupts occur during one PWM period. If a very short pulse is needed, the interrupt service routine will generate this short pulse using method B. Afterwards it writes a new number corresponding to the next long pulse to the timer. Here, only one interrupt occurs during one PWM period. Knowing the computing time of each part of the interrupt service routine ensures that a change in methods will not visibly affect the PWM signal or vary the PWM frequency.

Now, let us have a look at a simple application of this PWM software module. The circuit diagram is shown in Figure 1.

Potentiometer R1 adjusts voltage from 0V to VDD. A PIC16C71 converts this voltage to an 8-bit value thirty times per second and sends it to PORTB. The eight LEDs connected to PORTB show the result. The PWM signal value, which is output on pin RA3, is shown by the intensity of the connected LED. If the value is equal to 0, a continuous low signal is generated. If the value is equal to 255 a continuous high signal is generated.

The method of operation of the program is the same as the described procedure, which is shown by Table 1.

The file register `COUNTER` is the file register of the system clock. It is incremented by the PWM module at the rising edge of the PWM signal. Bit 7 is toggled each 255 • 128 • TCY and is used as the system clock bit.

The file register `PWMDESIRED` contains the desired PWM value in the range of 0 to 255. Register `PWMMAX` and `PWMHELP` are needed by the PWM module. They must not be modified by any other part of the program except during initialization.

The constant `PWMADJUSTVAL` contains the previously described factors, which have to be taken into account while calculating the adjusted timer value. The constant `PWMMAXVAL` controls the maximum pulse length, which is generated by making use of the time needed to execute `BSF`/`BCF` instructions. If the interrupt service routine is modified, then both constants will also have to be modified.

The comments in the code explain each instruction sequence.

CONCLUSION

This software based PWM module is able to generate a PWM frequency of up to 19.6 kHz with a variable duty cycle of 0% to 100%. The consumption of RAM is 3 bytes, the consumption of ROM less than 10% (nearly 100 instructions) and the consumption of computing time, in the worst case, amounts to 57/266 = 22.4%. The remaining computing time is more than the total available computing time of a 8051 microcontroller with a 12 MHz crystal.

© 1997 Microchip Technology Inc.

FIGURE 1:

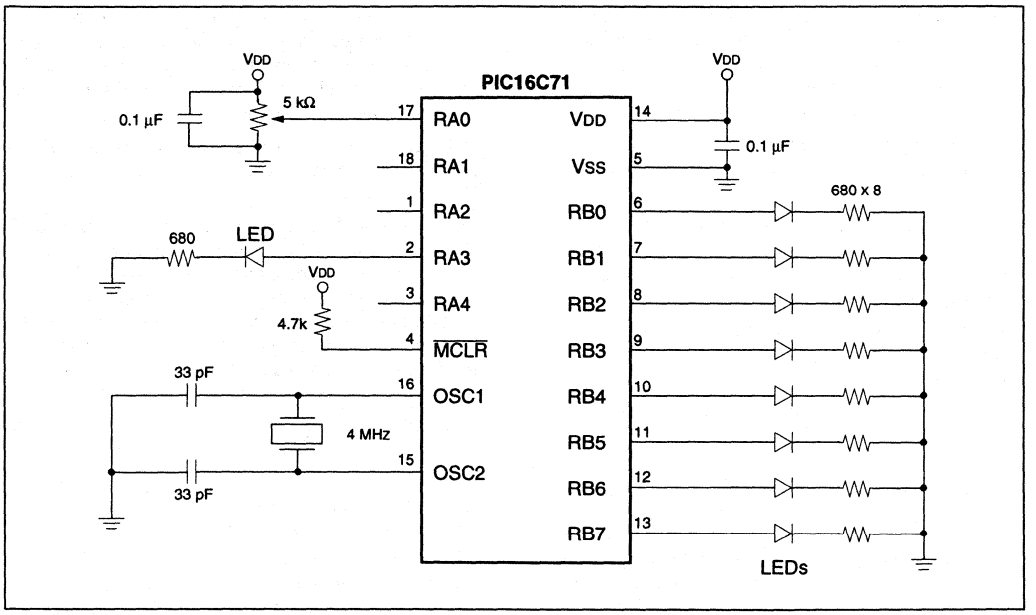


TABLE 1: OPERATION MODES OF THE SOFTWARE PWM MODULE

Duty Cycle Range	High Pulse	Low Pulse
0% .. 10%	Skipping (Method B)	Timer Interrupt
10% .. 90%	Timer Interrupt	Timer Interrupt
90% .. 100%	Timer Interrupt	Skipping (Method B)

© 1997 Microchip Technology Inc.

APPENDIX A:

```
MPASM 01.40 Released              PWM.ASM   10-2-1996  10:29:19         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;*******************************************************************
                     00002 ; Filename:  PWM16CXX.ASM
                     00003 ;*******************************************************************
                     00004 ;     Author:       Ole Ropcke
                     00005 ;     Company:      private
                     00006 ;     Revision:     RevA1
                     00007 ;     Date:         01-4-96
                     00008 ;     Assembled using MPASM rev 01.21
                     00009 ;*******************************************************************
                     00010 ;     Include files:
                     00011 ;       p16c71.inc
                     00012 ;*******************************************************************
                     00013 ;     Required hardware:
                     00014 ;*******************************************************************
                     00015 ;     What's changed:
                     00016 ; Date      Description of Change
                     00017 ; xx-xx-xx
                     00018 ;*******************************************************************
                     00019
                     00020  LIST P=16C71, R=DEC
                     00021
                     00022  include P16c71.INC
                     00001         LIST
                     00002 ; P16C71.INC  Std Hdr File, Version 1.00 Microchip Technology, Inc.
                     00142         LIST
                     00023
                     00024 ;*******************************************************************
                     00025 ; Definitions
                     00026 ;*******************************************************************
                     00027 ;
                     00028 ; I/O, Interrupt and Option Definitions
                     00029 ;
  00000088           00030 OPTIONVAL         equ   10001000b   ; portB no pull-up, tmr0 int.
  000000A0           00031 INTCONVAL         equ   0a0h        ; set GIE, T0IE
                     00032                                       ; port A:
  00000000           00033 UINPBIT           equ   00h         ; analog input for desired PWM
  00000003           00034 PWMOUTBIT         equ   03h         ; PWM output
  00000017           00035 TRISAVAL          equ   00010111b   ; A3 output
  00000002           00036 ADCON1VAL         equ   2           ; A0,A1 analog; A2,A3 digital
  00000089           00037 ADCON0VAL         equ   10001001b   ; fosc/32, channel 0
                     00038                                     ; port B:
  00000000           00039 TRISBVAL          equ   0           ; LED outputs
                     00040 ;-------------------------------------------------------------------
                     00041 ; Register Definitions
                     00042 ;
  0000000C           00043 STACKW        equ     0ch     ; stack to push/pop the W-register
  0000000D           00044 STACKS        equ     0dh     ; stack to push/pop the STATUS-reg
  0000000E           00045 COUNTER       equ     0eh     ; counter: input frequency
                     00046                               ;   f1 = crystalfreq. / 4 / 255
  0000000F           00047 COUNTER2      equ     0fh     ; counter2: input frequency
                     00048                               ;          f2 = f1 / 128
  00000010           00049 PWMDESIRED    equ     10h     ; desired PWM value 0..255
  00000011           00050 PWMMAX        equ     11h     ; register to support generating PWM
                     00051                               ; You have to put PWMMAXVAL into it!
  00000012           00052 PWMHELP       equ     12h     ; register to support generating PWM
                     00053                               ; used as temp storage of PWMDESIRED
                     00054 ;-------------------------------------------------------------------
                     00055 ; PWM-module-constant
  00000016           00056 PWMADJUSTVAL  equ     .22
```


© 1997 Microchip Technology Inc.

```
                    00057      ; correction number, defined by the following factors:
                    00058      ; time from timer interrupt to executing PC 004 + 3 cycles
                    00059      ; computing time from PC=004 to required edge   +18 cycles
                    00060      ; lost timer cycles due to writing the timer    + 2 cycles
                    00061      ; cal desired PWM value to timer loading value  + 2 cycles
                    00062      ; time from timer loading to gen required edge  - 1 cycle
                    00063      ; valid value for hardware (unknown diff to the data sheet)
                    00064      ; 3+18+0+2-1=22
                    00065      ; valid value for PICSIM version 5.11 (error of PICSIM):
                    00066      ; 0+18+2+2-1=21
  0000001D          00067 PWMMAXVAL       equ     .29
                    00068      ; loading value for PWMMAX
                    00069      ; If n is the maximum length of a high pulse, which has to be
                    00070      ; generated by the skipping method, then is PWMMAXVAL = n+1.
                    00071      ; The max length of a low pulse using the skip method is n-1.
                    00072 ;=================================================================
0000                00073          ORG     0
0000 2870           00074          goto    PowerOn
                    00075 ;--- PWM-Generator ------------------------------------------------
0004                00076          ORG     04h                ; timer interrupt
0004 1801           00077          btfsc   TMR0,0             ; compensate comp time of 1/2 cyc
0005 2806           00078          goto    PwmInt             ; instruc when timer int occured
0006                00079 PwmInt
0006 008C           00080          movwf   STACKW             ; copy W register... to "stack"
0007 0E8C           00081          swapf   STACKW,F           ; ...to "stack"
0008 0E03           00082          swapf   STATUS,W           ; copy STATUS register...
0009 008D           00083          movwf   STACKS             ; ...to "stack"
000A 110B           00084          bcf     INTCON,T0IF        ; clear interrupt flag
000B 1985           00085          btfsc   PORTA,PWMOUTBIT    ; which edge is required?
000C 2840           00086          goto    Lowpulse           ; -> goto falling edge
000D                00087 Highpulse
000D 0910           00088          comf    PWMDESIRED,W       ; get desired PWM value
000E 0092           00089          movwf   PWMHELP            ; store val for the foll low pulse
000F 0791           00090          addwf   PWMMAX,F           ; calc number of inst's to skip
0010 1C03           00091          btfss   STATUS,C           ; which method to use?
0011 2836           00092          goto    HighImpInt         ; -> using interrupt
0012                00093 HighImpShrt
0012 0811           00094          movf    PWMMAX,W           ; get number of inst's to skip
0013 0782           00095          addwf   PCL,F              ; skip n instructions
0014 1585           00096          bsf     PORTA,PWMOUTBIT    ; rising edge, 28 cycles hi pulse
0015 1585           00097          bsf     PORTA,PWMOUTBIT    ; 27 cycles
0016 1585           00098          bsf     PORTA,PWMOUTBIT    ; 26 cycles
0017 1585           00099          bsf     PORTA,PWMOUTBIT    ; 25 cycles
0018 1585           00100          bsf     PORTA,PWMOUTBIT    ; 24 cycles
0019 1585           00101          bsf     PORTA,PWMOUTBIT    ; 23 cycles
001A 1585           00102          bsf     PORTA,PWMOUTBIT    ; 22 cycles
001B 1585           00103          bsf     PORTA,PWMOUTBIT    ; 21 cycles
001C 1585           00104          bsf     PORTA,PWMOUTBIT    ; 20 cycles
001D 1585           00105          bsf     PORTA,PWMOUTBIT    ; 19 cycles
001E 1585           00106          bsf     PORTA,PWMOUTBIT    ; 18 cycles
001F 1585           00107          bsf     PORTA,PWMOUTBIT    ; 17 cycles
0020 1585           00108          bsf     PORTA,PWMOUTBIT    ; 16 cycles
0021 1585           00109          bsf     PORTA,PWMOUTBIT    ; 15 cycles
0022 1585           00110          bsf     PORTA,PWMOUTBIT    ; 14 cycles
0023 1585           00111          bsf     PORTA,PWMOUTBIT    ; 13 cycles
0024 1585           00112          bsf     PORTA,PWMOUTBIT    ; 12 cycles
0025 1585           00113          bsf     PORTA,PWMOUTBIT    ; 11 cycles
0026 1585           00114          bsf     PORTA,PWMOUTBIT    ; 10 cycles
0027 1585           00115          bsf     PORTA,PWMOUTBIT    ;  9 cycles
0028 1585           00116          bsf     PORTA,PWMOUTBIT    ;  8 cycles
0029 1585           00117          bsf     PORTA,PWMOUTBIT    ;  7 cycles
002A 1585           00118          bsf     PORTA,PWMOUTBIT    ;  6 cycles
002B 1585           00119          bsf     PORTA,PWMOUTBIT    ;  5 cycles
002C 1585           00120          bsf     PORTA,PWMOUTBIT    ;  4 cycles
002D 1585           00121          bsf     PORTA,PWMOUTBIT    ;  3 cycles
002E 1585           00122          bsf     PORTA,PWMOUTBIT    ;  2 cycles
```


© 1997 Microchip Technology Inc.

```
002F 1585           00123          bsf     PORTA,PWMOUTBIT  ;  1 cycle
0030 1185           00124          bcf     PORTA,PWMOUTBIT  ; fall edge;start of the following
                    00125                                   ; low pulse using the interrupt
0031 0A8E           00126          incf    COUNTER,F        ; trigger COUNTER, cause there was
                    00127                                   ; a rising edge
0032 0912           00128          comf    PWMHELP,W        ; get required low pulse length
0033 3E1B           00129          addlw   PWMADJUSTVAL+5   ; calculate timer loading value
                    00130                                   ; Edge was generated 5 cycles before
                    00131                                   ; usual point of time.
0034 0081           00132          movwf   TMR0             ; put value into timer
0035 2869           00133          goto    LowImpInt2       ; low pulse using int is running
0036                00134 HighImpInt                        ; high pulse using interrupt
0036 3E16           00135          addlw   PWMADJUSTVAL     ; calculate timer loading value
0037 0081           00136          movwf   TMR0             ; put value into timer
0038                00137 HighImpInt2
0038 1585           00138          bsf     PORTA,PWMOUTBIT  ; generate rising edge
0039 0A8E           00139          incf    COUNTER,F        ; trigger counter, because there
                    00140                                   ; was a rising edge
003A 301C           00141          movlw   PWMMAXVAL-1      ; "repair"...
003B 0091           00142          movwf   PWMMAX           ; ...support register
003C 0E0D           00143          swapf   STACKS,W         ; restore...
003D 0083           00144          movwf   STATUS           ; ...STATUS register
003E 0E0C           00145          swapf   STACKW,W         ; restore W register
003F 0009           00146          retfie                   ; return to main program
0040                00147 Lowpulse
0040 0912           00148          comf    PWMHELP,W        ; get required pulse length
0041 0791           00149          addwf   PWMMAX,F         ; calc number of inst's to skip
0042 1C03           00150          btfss   STATUS,C         ; which method is to use?
0043 2867           00151          goto    LowImpInt        ; ->using interrupt
0044                00152 LowImpShrt
0044 0811           00153          movf    PWMMAX,W         ; get number of inst's to skip
0045 0782           00154          addwf   PCL,F            ; skip n instructions
0046 1185           00155          bcf     PORTA,PWMOUTBIT  ; falling edge, 27 cycles low pulse
0047 1185           00156          bcf     PORTA,PWMOUTBIT  ; 26 cycles
0048 1185           00157          bcf     PORTA,PWMOUTBIT  ; 25 cycles
0049 1185           00158          bcf     PORTA,PWMOUTBIT  ; 24 cycles
004A 1185           00159          bcf     PORTA,PWMOUTBIT  ; 23 cycles
004B 1185           00160          bcf     PORTA,PWMOUTBIT  ; 22 cycles
004C 1185           00161          bcf     PORTA,PWMOUTBIT  ; 21 cycles
004D 1185           00162          bcf     PORTA,PWMOUTBIT  ; 20 cycles
004E 1185           00163          bcf     PORTA,PWMOUTBIT  ; 19 cycles
004F 1185           00164          bcf     PORTA,PWMOUTBIT  ; 18 cycles
0050 1185           00165          bcf     PORTA,PWMOUTBIT  ; 17 cycles
0051 1185           00166          bcf     PORTA,PWMOUTBIT  ; 16 cycles
0052 1185           00167          bcf     PORTA,PWMOUTBIT  ; 15 cycles
0053 1185           00168          bcf     PORTA,PWMOUTBIT  ; 14 cycles
0054 1185           00169          bcf     PORTA,PWMOUTBIT  ; 13 cycles
0055 1185           00170          bcf     PORTA,PWMOUTBIT  ; 12 cycles
0056 1185           00171          bcf     PORTA,PWMOUTBIT  ; 11 cycles
0057 1185           00172          bcf     PORTA,PWMOUTBIT  ; 10 cycles
0058 1185           00173          bcf     PORTA,PWMOUTBIT  ;  9 cycles
0059 1185           00174          bcf     PORTA,PWMOUTBIT  ;  8 cycles
005A 1185           00175          bcf     PORTA,PWMOUTBIT  ;  7 cycles
005B 1185           00176          bcf     PORTA,PWMOUTBIT  ;  6 cycles
005C 1185           00177          bcf     PORTA,PWMOUTBIT  ;  5 cycles
005D 1185           00178          bcf     PORTA,PWMOUTBIT  ;  4 cycles
005E 1185           00179          bcf     PORTA,PWMOUTBIT  ;  3 cycles
005F 1185           00180          bcf     PORTA,PWMOUTBIT  ;  2 cycles
0060 1185           00181          bcf     PORTA,PWMOUTBIT  ;  1 cycle
0061 1585           00182          bsf     PORTA,PWMOUTBIT  ; rising edge; start of the next
                    00183                                   ; high pulse using the interrupt
0062 0910           00184          comf    PWMDESIRED,W     ; get desired PWM value
0063 0092           00185          movwf   PWMHELP          ; store val for the next lo pulse
0064 3E1B           00186          addlw   PWMADJUSTVAL+5   ; calculate timer loading value
                    00187                                   ; Edge was gen'd 5 cycles before
                    00188                                   ; usual point of time.
```

```
0065 0081           00189         movwf   TMR0              ; put value into timer
0066 2838           00190         goto    HighImpInt2       ; high pulse using int is running
0067                00191 LowImpInt                         ; low pulse using interrupt
0067 3E16           00192         addlw   PWMADJUSTVAL      ; calculate timer loading value
0068 0081           00193         movwf   TMR0              ; put value into timer
0069                00194 LowImpInt2
0069 1185           00195         bcf     PORTA,PWMOUTBIT   ; generate falling edge
006A 301D           00196         movlw   PWMMAXVAL         ; "repair" ...
006B 0091           00197         movwf   PWMMAX            ; ...support register
006C 0E0D           00198         swapf   STACKS,W          ; restore ...
006D 0083           00199         movwf   STATUS            ; ...STATUS register
006E 0E0C           00200         swapf   STACKW,W          ; restore W register
006F 0009           00201         retfie                    ; return to main program
                    00202 ;--- power on routine -----------------------------------------
0070                00203 PowerOn
                    00204         ; configuration of the PWM module
0070 0181           00205         clrf    TMR0              ; reset timer
0071 0190           00206         clrf    PWMDESIRED        ; reset value of PWM is 0
0072 1185           00207         bcf     PORTA,PWMOUTBIT   ; reset PWM-port before port A is
                    00208                                   ; changed from input to output to
                    00209                                   ; suppress an uncontrolled spike
0073 301D           00210         movlw   PWMMAXVAL         ; set support register
0074 0091           00211         movwf   PWMMAX            ;
                    00212         ; configuration of the Pic
0075 1683           00213         bsf     STATUS,RP0        ; register page 1
0076 3017           00214         movlw   TRISAVAL          ; configurate ...
0077 0085           00215         movwf   TRISA             ; ...port A
0078 3000           00216         movlw   TRISBVAL          ; configurate ...
0079 0086           00217         movwf   TRISB             ; ...port B
007A 3002           00218         movlw   ADCON1VAL         ; set inputs of ...
007B 0088           00219         movwf   ADCON1            ; ...adc
007C 3088           00220         movlw   OPTIONVAL         ; configurate ...
007D 0081           00221         movwf   OPTION_REG        ; ...Pic
007E 1283           00222         bcf     STATUS,RP0        ; register page 0
007F 30A0           00223         movlw   INTCONVAL         ; configure interrupts and ...
0080 008B           00224         movwf   INTCON            ; ...enable interrupts
                    00225 ;--- main idle ------------------------------------------------
0081                00226 Idle
0081 0064           00227         clrwdt                    ; toggle watchdog
0082 1F8E           00228         btfss   COUNTER,07h       ; if MSB isn't set, ...
0083 2881           00229         goto    Idle              ; ...then waiting
0084 0A8F           00230         incf    COUNTER2,F        ; ...else toggle COUNTER2
                    00231                                   ; If the crystal freq is 4 MHz,
                    00232                                   ; the toggle freq is nearly 30.6 Hz
0085 138E           00233         bcf     COUNTER,07h       ; reset MSB
0086 3089           00234         movlw   ADCON0VAL         ; set ...
0087 0088           00235         movwf   ADCON0            ; ...ADC configuration
0088                00236 WaitNoInt
0088 0801           00237         movf    TMR0,W            ; waiting until enough time
0089 3CD0           00238         sublw   0d0h              ; for one conversion before start
008A 1C03           00239         btfss   STATUS,C          ; of the next timer interrupt.
008B 2888           00240         goto    WaitNoInt         ; (Conv can be disturbed by
                    00241                                   ; an interrupt.
008C 1508           00242         bsf     ADCON0,GO         ; start ADC
008D                00243 WaitAdc
008D 1908           00244         btfsc   ADCON0,GO         ; waiting until ADC ...
008E 288D           00245         goto    WaitAdc           ; ... is ready
008F 0809           00246         movf    ADRES,W           ; put result into W-reg and then
0090 0090           00247         movwf   PWMDESIRED        ; desired PWM-value into PWMDESIRED
0091 0086           00248         movwf   PORTB             ; show result at port B (LEDs)
0092 2881           00249         goto    Idle              ;
                    00250 ;-----------------------------------------------------------------
                    00251         END
```


© 1997 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXX------------ ---------------- ----------------

All other memory blocks unused.

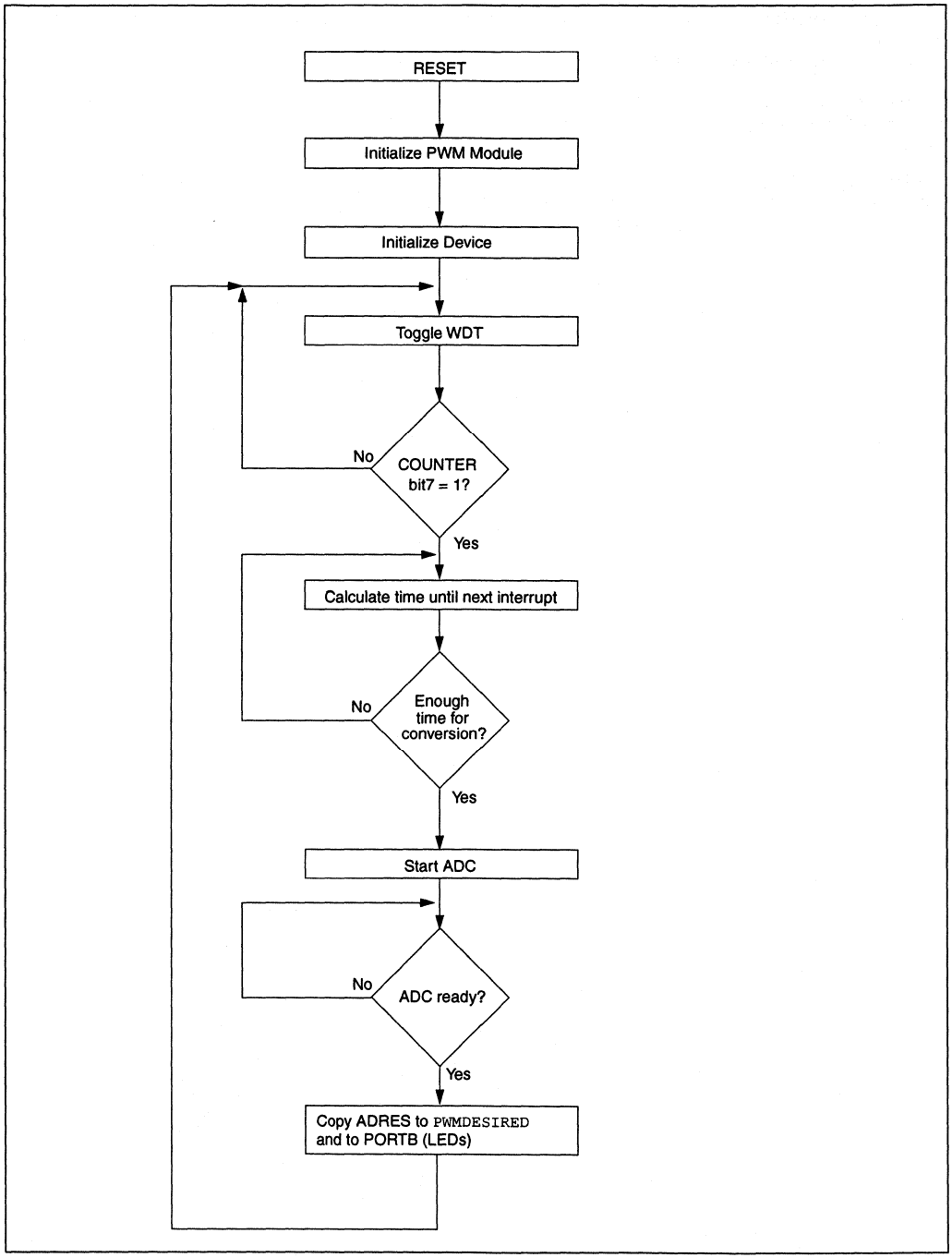
Program Memory Words Used:   144
Program Memory Words Free:   880


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

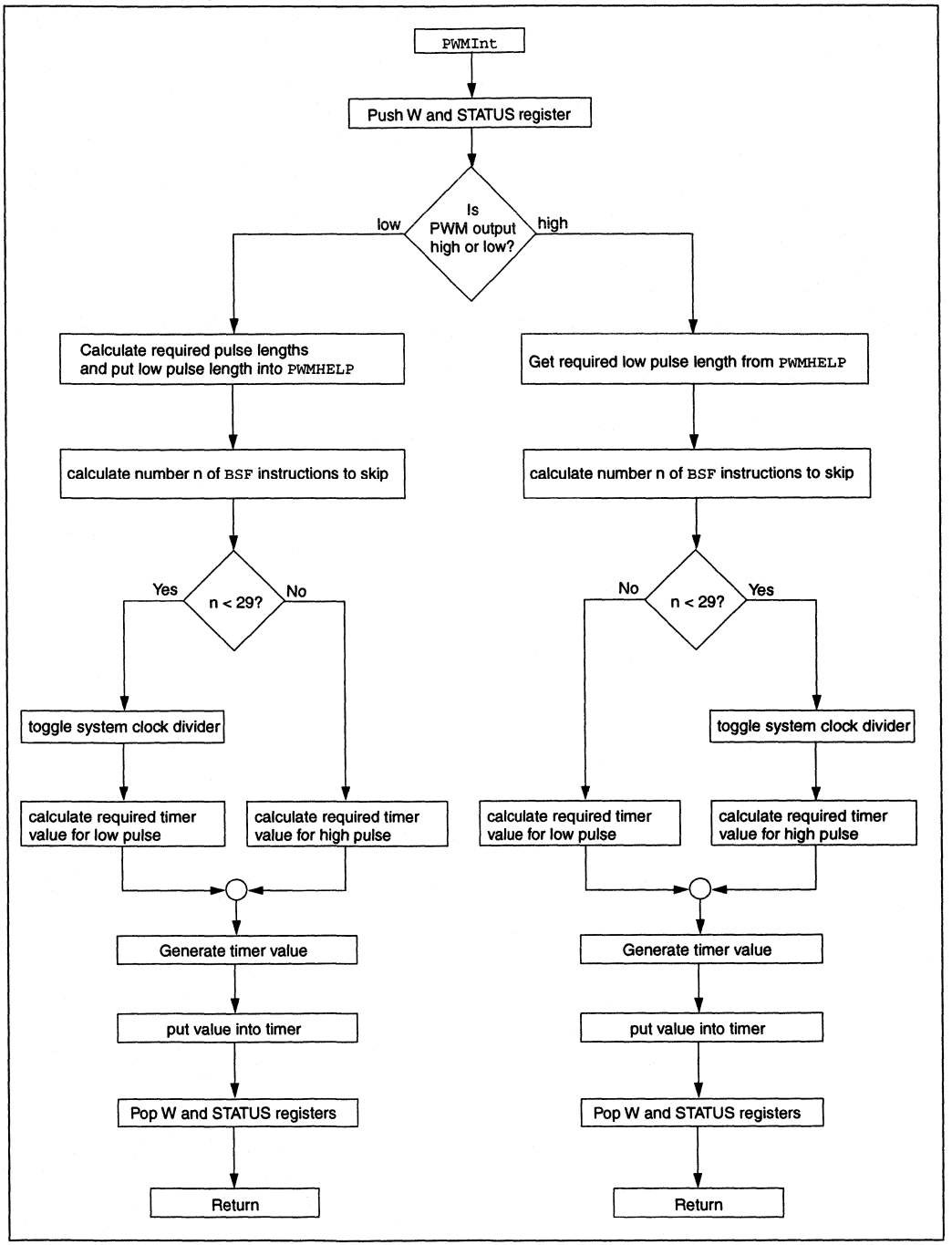
APPENDIX B: FLOWCHARTS

FIGURE B-1: MAIN ROUTINE



© 1997 Microchip Technology Inc.

FIGURE B-2: PWM GENERATOR ROUTINE



© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

AN656

In-Circuit Serial Programming of Calibration Parameters Using a PICmicro™ Microcontroller

Author: John Day
Microchip Technology Inc.

INTRODUCTION

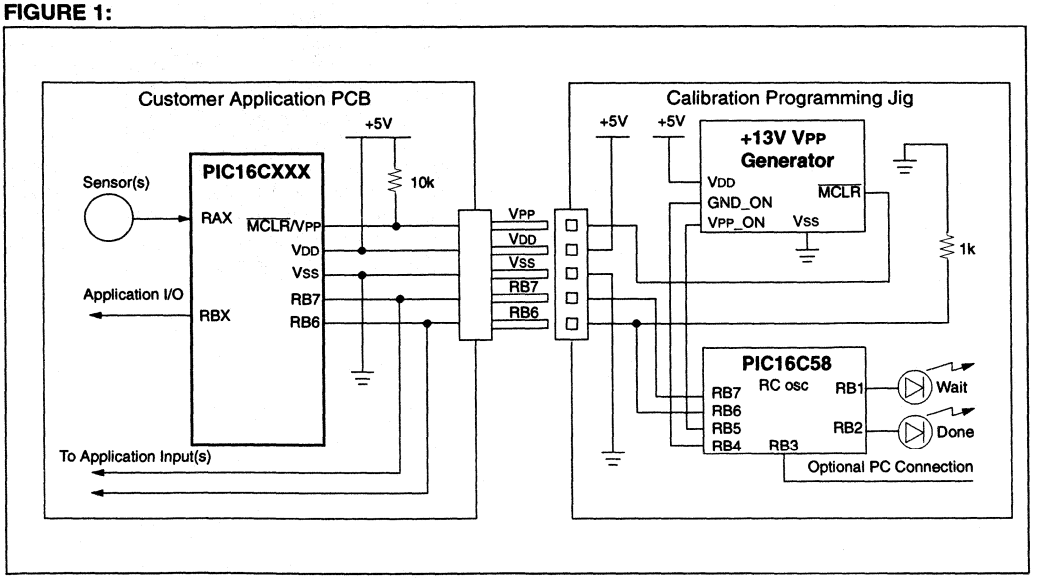
Many embedded control applications, where sensor offsets, slopes and configuration information are measured and stored, require a calibration step. Traditionally, potentiometers or serial EEPROM devices are used to set up and store this calibration information. This application note will show how to construct a programming jig that will receive calibration parameters from the application midrange PICmicro and program this information into the application baseline PICmicro using the In-Circuit Programming protocol. This method uses the PIC16CXXX In Circuit Serial Programming algorithm of the 14-bit core microcontrollers.

PROGRAMMING FIXTURE

A programming fixture is needed to assist with the self programming operation. This is typically a small re-usable module that plugs into the application PCB being calibrated. Only 5 pin connections are needed and this programming fixture can draw its power from the application PCB to simplify the connections.

3

FIGURE 1:



© 1997 Microchip Technology Inc.

Electrical Interface

There are a total of five electrical connections needed between the application PIC16CXXX microcontroller and the programming jig:

- **$\overline{MCLR}$/VPP** - High voltage pin used to place application PIC16CXX into programming mode
- **VDD** - +5 volt power supply connection to the application PIC16CXXX
- **Vss** - Ground power supply connection to the application PIC16CXXX
- **RB6** - PORTB, bit6 connection to application PIC16CXX used to clock programming data
- **RB7** - PORTB, bit7 connection to application PIC16CXX used to send programming data

This programming jig is intended to grab power from the application power supply through the VDD connection. The programming jig will require 100 mA of peak current during programming. The application will need to set RB6 and RB7 as inputs, which means external devices cannot drive these lines. The calibration data will be sent to the programming jig by the application PIC16CXXX through RB6 and RB7. The programming jig will later use these lines to clock the calibration data into the application PIC16CXXX.

Programming Issues

The PIC16CXXX Programming specification suggests verification of program memory Specification at both Maximum and Minimum VDD for each device. This is done to ensure proper programming margins and to detect (and reject) any improperly programmed devices. All production quality programmers vary VDD from VDD min to VDD max after programming and verify the device under each of these conditions.

Since both the application voltage and it's tolerances are known, it is not necessary to verify the PIC16CXXX calibration parameters at the device VDD Max and VDD Min. It is only necessary to verify at the application power supply Max and Min voltages. This application note shows the nominal (+5V) verification routine and hardware. If the power supply is a regulated +5V, this is adequate and no additional hardware or software is needed. If the application power supply is not regulated (such as a battery powered or poorly regulated system) it is important to complete a VDD min and VDD Max verification cycle following the +5V verification cycle. See programming specifications for more details on VDD verification procedures.

- PIC16C5X Programming Specifications - DS30190
- PIC16C55X Programming Specifications - DS30261
- PIC16C6X/7X/9XX Programming Specifications - DS30228
- PIC16C84 Programming Specifications - DS30189

Note: The designer must consider environmental conditions, voltage ranges, and aging issues when determing VDD min/max verification levels. Please refer to the programming specification for the application device.

The calibration programming and initial verification MUST occur at +5V. If the application is intended to run at lower (or higher voltage), a second verification pass must be added where those voltages are applied to VDD and the device is verified.

© 1997 Microchip Technology Inc.

Communication format (Application Microcontroller to Programming Jig)

Unused program memory, in the application PIC16CXXX, is left unprogrammed as all 1s; therefore the unprogrammed program memory for the calibration look-up table would contain 3FFF (hex). This is interpreted as a "ADDLW FF". The application microcontroller simply needs one "RETLW FF" instruction at the end of the space allocated in program memory for the calibration parameter look-up table. When the application microcontroller is powered up, it will receive a "FFh" for each calibration parameter that is looked up; therefore, it can detect that it is uncalibrated and jump to the calibration code.

Once the calibration constants are calculated by the application PICmicro, they need to be communicated to the (PIC16C58A based) programming jig. This is accomplished through the RB6 and RB7 lines. The format is a simple synchronous clock and data format as shown in Figure 2.

A pulldown on the clock line is used to hold it low. The application microcontroller needs to send the high and low bytes of the target start address of the calibration constants to the calibration jig. Next, the data bytes are sent followed by a checksum of the entire data transfer as shown in Figure 3.

Once the data transfer is complete, the checksum is verified by the programming jig and the data printed at 9600 baud, 8-bits, no parity, 1 stop bit through RB3. A connection to this pin is optional. Next the programming jig applies +13V, programs and verifies the application PIC16CXXX calibration parameters.

3

FIGURE 2:

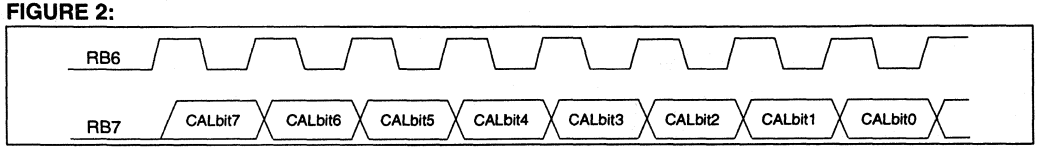
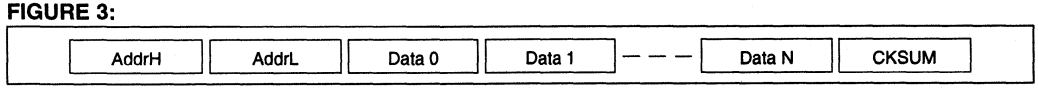


FIGURE 3:

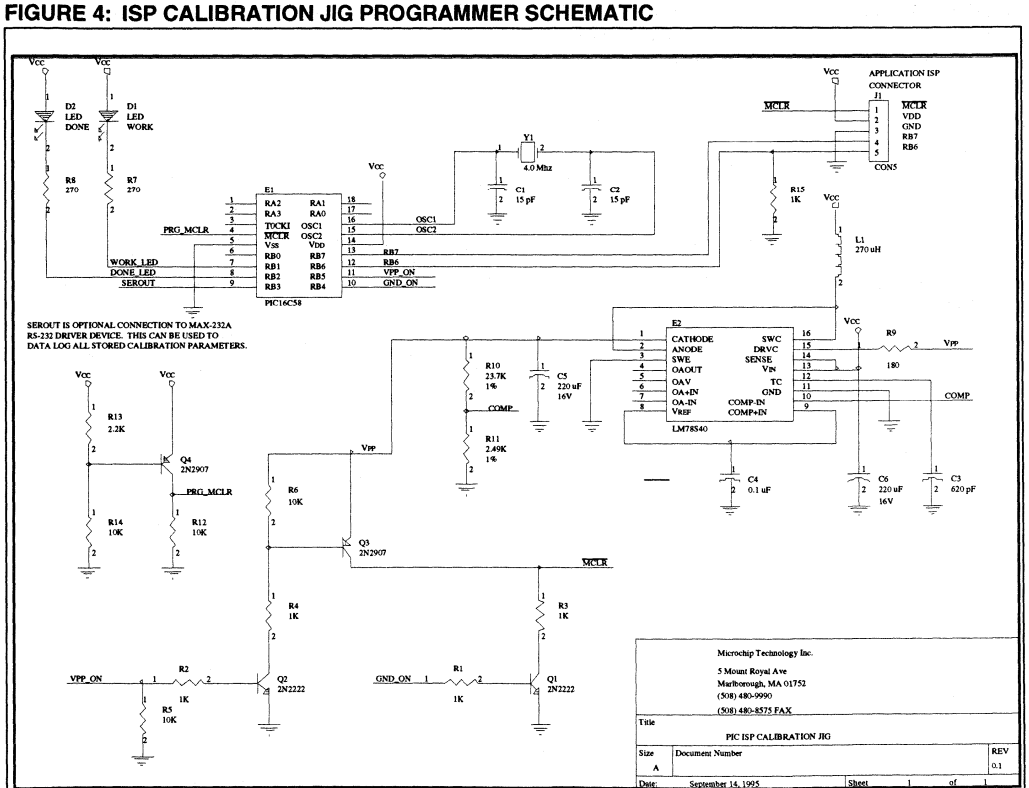


© 1997 Microchip Technology Inc.

LED Operation

When the programming jig is waiting for communication from the application PICmicro, both LEDs are OFF. Once a valid data stream is received (with at least one calibration byte and a correct checksum) the WORK LED is lit while the calibration parameters are printed through the optional RB3 port. Next, the DONE LED is lit to indicate that these parameters are being programmed and verified by the programming jig. Once the programming is finished, the WORK LED is extinguished and the DONE LED remains lit. If any parameters fail programming, the DONE LED is extinguished; therefore both LEDs would remain off.

FIGURE 4: ISP CALIBRATION JIG PROGRAMMER SCHEMATIC



© 1997 Microchip Technology Inc.

Code Protection

Selection of the code protection configuration bit on PIC16CXXX microcontrollers prevents further programming of the program memory array. This would prevent writing self calibration parameters if the device is code protected prior to calibration. There are two ways to address this issue:

1. Do not code protect the device when programming it with the programmer. Add additional code (See the PIC16C6X/7X programming Spec) to the `ISPPRGM.ASM` to program the code protection bit after complete verification of the calibration parameters
2. Only code protect 1/2 or 3/4 of the program memory with the programmer. Place the calibration constants into the unprotected part of program memory.

Software Routines

There are two source code files needed for this application note:

1. `ISPTEST.ASM` (Appendix A) Contains the source code for the application PIC16CXXX, sets up the calibration look-up table and implements the communication protocol to the programming jig.

2. `ISPPRGM.ASM` (Appendix B) Source code for a PIC16C58A to implement the programming jig. This waits for and receives the calibration parameters from the application PIC16CXXX, places it into programming mode and programs/verifies each calibration word.

CONCLUSION

Typically, calibration information about a system is stored in EEPROM. For calibration data that does not change over time, the In-circuit Serial Programming capability of the PIC16CXXX devices provide a simple, cost effective solution to an external EEPROM. This method not only decreases the cost of a design, but also reduces the complexity and possible failure points of the application.

3

TABLE 1: PARTS LIST FOR PIC16CXXX ISP CALIBRATION JIG

Bill of Material			
Item	**Quantity**	**Reference**	**Part**
1	2	C1,C2	15 pF
2	1	C3	620 pF
3	1	C4	0.1 mF
4	2	C5,C6	220 mF
5	2	D1,D2	LED
6	1	E1	PIC16C58
7	1	E2	LM78S40
8	1	J1	CON5
9	1	L1	270 mH
10	2	Q1,Q2	2N2222
11	2	Q3,Q4	2N2907
12	5	R1,R2,R3,R4,R15	1k
13	4	R5,R6,R12,R14	10k
14	2	R7,R8	270
15	1	R9	180
16	1	R10	23.7k
17	1	R11	2.49k
18	1	R13	2.2k
19	1	Y1	4.0 MHz

© 1997 Microchip Technology Inc.

APPENDIX A:

```
MPASM 01.40.01 Intermediate   ISPPRGM.ASM   3-31-1997  10:57:03         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ; Filename: ISPPRGM.ASM
                     00002 ; ********************************************
                     00003 ; * Author:   John Day                       *
                     00004 ; *           Sr. Field Applications Engineer  *
                     00005 ; *           Microchip Technology             *
                     00006 ; * Revision: 1.0                            *
                     00007 ; * Date      August 25, 1995                *
                     00008 ; * Part:     PIC16C58                       *
                     00009 ; * Compiled using MPASM V1.40               *
                     00010 ; ********************************************
                     00011 ; * Include files:                           *
                     00012 ; *           P16C5X.ASM                     *
                     00013 ; ********************************************
                     00014 ; * Fuses:    OSC:  XT (4.0 Mhz xtal)        *
                     00015 ; *           WDT:  OFF                      *
                     00016 ; *            CP:  OFF                      *
                     00017
                     ;*******************************************************************************
                     00018 ; This program is intended to be used as a self programmer
                     00019 ; to store calibration constants into a lookup table
                     00020 ; within the main system processor.  A 4 Mhz crystal
                     00021 ; is needed and an optional 9600 baud seiral port will
                     00022 ; display the parameters to be programmed.
                     00023 ;
                     ;*******************************************************************************
                     00024 ; * Program Memory:                                                      *
                     00025 ; *    Words - communication with test jig                              *
                     00026 ; *    17 Words - calibration look-up table (16 bytes of data)          *
                     00027 ; *    13 Words - Test Code to generate Calibration Constants           *
                     00028 ; * RAM memory:                                                         *
                     00029 ; *    64 Bytes - Store up to 64 bytes of calibration constant          *
                     00030 ; *     9 Bytes - Store 9 bytes of temp variables (reused)              *
                     00031 ;
                     ;*******************************************************************************
                     00032
                     00033     list p=16C58A
                     00034     include <p16C5x.inc>
                     00001         LIST
                     00002 ; P16C5X.INC Standard Hdr File, Version 3.30 Microchip Technology, Inc.
                     00224         LIST
0FFF 0FF9            00035     __CONFIG _CP_OFF&_WDT_OFF&_XT_OSC
                     00036
                     00037 ; ***********************************
                     00038 ; * Port A (RA0-RA4) bit definitions *
                     00039 ; ***********************************
                     00040 ; No PORT A pins are used in this design
                     00041
                     00042 ; ***********************************
                     00043 ; * Port B (RB0-RB7) bit definitions *
                     00044 ; ***********************************
  00000006           00045 ISPCLOCK      EQU 6   ; Clock line for ISP and parameter comm
  00000007           00046 ISPDATA       EQU 7   ; Data line for ISP and parameter comm
  00000005           00047 VPPON         EQU 5   ; Apply +13V VPP voltage to MCLR (test mode)
  00000004           00048 GNDON         EQU 4   ; Apply +0V (gnd) voltage to MCLR (reset)
  00000003           00049 SEROUT        EQU 3   ; Optional RS-232 TX output (needs 12V driver)
  00000002           00050 DONELED       EQU 2   ; Turns on LED when done sucessfully program
  00000001           00051 WORKLED       EQU 1   ; On during programming, off when done
                     00052                       ; RB0 is not used in this design
                     00053
```


© 1997 Microchip Technology Inc.

```
                00054 ; ************************************************
                00055 ; * RAM register definition:                      *
                00056 ; *  07h - 0Fh - used for internal counters, vars *
                00057 ; *  10h - 7Fh - 64 bytes for cal param storage   *
                00058 ; ************************************************
                00059 ; ***
                00060 ; *** The following VARS are used during ISP programming:
                00061 ; ***
00000007        00062 HIADDR          EQU 07h  ; High address of CAL params to be stored
00000008        00063 LOADDR          EQU 08h  ; Low address of CAL params to be stored
00000007        00064 HIDATA          EQU 07h  ; High byte of data to be sent via ISP
00000008        00065 LODATA          EQU 08h  ; Low byte of data to be sent via ISP
00000009        00066 HIBYTE          EQU 09h  ; High byte of data received via ISP
0000000A        00067 LOBYTE          EQU 0Ah  ; Low byte of data received via ISP
0000000B        00068 PULSECNT        EQU 0Bh  ; Number of times PIC has been pulse programmed
0000000C        00069 TEMPCOUNT       EQU 0Ch  ; TEMP var used in counters
0000000D        00070 TEMP            EQU 0Dh  ; TEMP var used throughout program
                00071 ; ***
                00072 ; *** The following VARS are used to receive and store CAL params:
                00073 ; ***
00000007        00074 COUNT           EQU 07h  ; Counter var used to receive cal params
00000008        00075 TEMP1           EQU 08h  ; TEMP var used for RS-232 comm
00000009        00076 DATAREG         EQU 09h  ; Data register used for RS-232 comm
0000000A        00077 CSUMTOTAL       EQU 0Ah  ; Running total of checksum (addr + data)
0000000B        00078 TIMEHIGH        EQU 0Bh  ; Count how long CLOCK line is high
0000000C        00079 TIMELOW         EQU 0Ch  ; Count how long CLOCK line is low
0000000E        00080 ADDRPTR         EQU 0Eh  ; Pointer to next byte of CAL storage
0000000F        00081 BYTECOUNT       EQU 0Fh  ; Number of CAL bytes received
                00082
                00083 ; ************************************
                00084 ; * Various constants used in program *
                00085 ; ************************************
00000001        00086 DATISPOUT       EQU b'00000001'  ; tris settings for ISP data out
00000081        00087 DATISPIN        EQU b'10000001'  ; tris settings for ISP data in
00000006        00088 CMDISPCNT       EQU 6            ; Number of bits for ISP command
00000010        00089 STARTCALBYTE    EQU 10h          ; Address in RAM where CAL byte data stored
00000007        00090 VFYYES          EQU PA2          ; Flag bit enables verification (STATUS)
00000006        00091 CMDISPINCRADDR EQU b'00000110'  ; ISP Pattern to increment address
00000008        00092 CMDISPPGMSTART EQU b'00001000'  ; ISP Pattern to start programming
0000000E        00093 CMDISPPGMEND    EQU b'00001110'  ; ISP Pattern to end programming
00000002        00094 CMDISPLOAD      EQU b'00000010'  ; ISP Pattern to load data for program
00000004        00095 CMDISPREAD      EQU b'00000100'  ; ISP Pattern to read data for verify
00000034        00096 UPPER6BITS      EQU 034h         ; Upper 6 bits for retlw instruction
                00097
                00098 ; ************************************
                00099 ; * delaybit macro                    *
                00100 ; * Delays for 104 uS (at 4 Mhz clock)*
                00101 ; * for 9600 baud communications      *
                00102 ; *     RAM used:       COUNT         *
                00103 ; ************************************
                00104 delaybit macro
                00105     local dlylabels
                00106 ; 9600 baud, 8 bit, no parity, 104 us per bit, 52 uS per half bit
                00107 ; (8) shift/usage + (2) setup + (1) nop + (3 * 31) literal = (104) 4Mhz
                00108     movlw   .31             ; place 31 decimal literal into count
                00109     movwf   COUNT           ; Initialize COUNT with loop count
                00110     nop                     ; Add one cycle delay
                00111 dlylabels
                00112     decfsz  COUNT,F         ; Decrement count until done
                00113     goto    dlylabels       ; Not done delaying - go back!
                00114     ENDM                    ; Done with Macro
                00115
                00116 ; ************************************************
                00117 ; * addrtofsr macro                             *
                00118 ; * Converts logical, continuous address 10h-4Fh *
                00119 ; * to FSR address as follows for access to (4)  *
```

```
                00120 ; * banks of file registers in PIC16C58:          *
                00121 ; *       Logical Address       FSR Value        *
                00122 ; *       10h-1Fh               10h-1Fh          *
                00123 ; *       20h-2Fh               30h-3Fh          *
                00124 ; *       30h-3Fh               50h-5Fh          *
                00125 ; *       40h-4Fh               70h-7Fh          *
                00126 ; *     Variable Passed: Logical Address         *
                00127 ; *     RAM used:        FSR                     *
                00128 ; *                      W                       *
                00129 ; ************************************************
                00130 addrtofsr macro TESTADDR
                00131     movlw   STARTCALBYTE        ; Place base address into W
                00132     subwf   TESTADDR,w          ; Offset by STARTCALBYTE
                00133     movwf   FSR                 ; Place into FSR
                00134     btfsc   FSR,5               ; Shift bits 4,5 to 5,6
                00135     bsf     FSR,6
                00136     bcf     FSR,5
                00137     btfsc   FSR,4
                00138     bsf     FSR,5
                00139     bsf     FSR,4
                00140     endm
                00141.
                00142
                00143 ; ************************************
                00144 ; * The PC starts at the END of memory *
                00145 ; ************************************
07FF            00146     ORG     7FFh
Message[306]: Crossing page boundary -- ensure page bits are set.
07FF 0A00       00147     goto    start
                00148
                00149 ; ************************************
                00150 ; * Start of CAL param read routine    *
                00151 ; ************************************
0000            00152     ORG     0h
0000            00153 start
0000 0C0A       00154     movlw   b'00001010'   ; Serial OFF, LEDS OFF, VPP OFF
0001 0026       00155     movwf   PORTB         ; Place "0" into port b latch register
0002 0CC1       00156     movlw   b'11000001'   ; RB7;:RB6, RB0 set to inputs
0003 0006       00157     tris    PORTB         ; Move to tris registers
0004 0040       00158     clrw                  ; Place 0 into W
0005 0065       00159     clrf    PORTA         ; Place all ZERO into latch
0006 0005       00160     tris    PORTA         ; Make all pins outputs to be safe..
0007 0586       00161     bsf     PORTB,GNDON   ; TEST ONLY-RESET PIC-NOT NEEDED IN REAL DESIGN!
0008            00162 clearram
0008 0C10       00163     movlw   010h          ; Place start of buffer into W
0009 0027       00164     movwf   COUNT         ; Use count for RAM pointer
000A            00165 loopclrram
                00166     addrtofsr COUNT       ; Set up FSR
000A 0C10           M     movlw   STARTCALBYTE  ; Place base address into W
000B 0087           M     subwf   COUNT,w       ; Offset by STARTCALBYTE
000C 0024           M     movwf   FSR           ; Place into FSR
000D 06A4           M     btfsc   FSR,5         ; Shift bits 4,5 to 5,6
000E 05C4           M     bsf     FSR,6
000F 04A4           M     bcf     FSR,5
0010 0684           M     btfsc   FSR,4
0011 05A4           M     bsf     FSR,5
0012 0584           M     bsf     FSR,4
0013 0060       00167     clrf    INDF          ; Clear buffer value
0014 02A7       00168     incf    COUNT,F       ; Move to next reg
0015 0C50       00169     movlw   050h          ; Move end of buffer addr to W
0016 0087       00170     subwf   COUNT,W       ; Check if at last MEM
0017 0743       00171     btfss   STATUS,Z      ; Skip when at end of counter
0018 0A0A       00172     goto    loopclrram    ; go back to next location
0019 0486       00173     bcf     PORTB,GNDON   ; TEST ONLY-LET IT GO-NOT NEEDED IN REAL DESIGN!
001A            00174 calget
001A 006A       00175     clrf    CSUMTOTAL     ; Clear checksum total byte
```


© 1997 Microchip Technology Inc.

```
001B 0069       00176      clrf    DATAREG             ; Clear out data receive register
001C 0C10       00177      movlw   STARTCALBYTE        ; Place RAM start address of first cal byte
001D 002E       00178      movwf   ADDRPTR             ; Place this into ADDRPTR
001E            00179 waitclockpulse
001E 07C6       00180      btfss   PORTB,ISPCLOCK      ; Wait for CLOCK high pulse - skip when high
001F 0A1E       00181      goto    waitclockpulse      ; CLOCK is low - go back and wait!
0020            00182 loopcal
0020 0C08       00183      movlw   .8                  ; Place 8 into W (8 bits/byte)
0021 0027       00184      movwf   COUNT               ; set up counter register to count bits
0022            00185 loopsendcal
0022 006B       00186      clrf    TIMEHIGH            ; Clear timeout counter for high pulse
0023 006C       00187      clrf    TIMELOW             ; Clear timeout counter for low pulse
0024            00188 waitclkhi
0024 06C6       00189      btfsc   PORTB,ISPCLOCK      ; Wait for CLOCK high - skip if it is low
0025 0A29       00190      goto    waitclklo           ; Jump to wait for CLOCK low state
0026 02EB       00191      decfsz  TIMEHIGH,F          ; Decrement counter - skip if timeout
0027 0A24       00192      goto    waitclkhi           ; Jump back and wait for CLOCK high again
0028 0A47       00193      goto    timeout             ; Timed out waiting for high - check data!
0029            00194 waitclklo
0029 07C6       00195      btfss   PORTB,ISPCLOCK      ; Wait for CLOCK low - skip if it is high
002A 0A2E       00196      goto    clockok             ; Got a high to low pulse - jump to clockok
002B 02EC       00197      decfsz  TIMELOW,F           ; Decrement counter - skip if timeout
002C 0A29       00198      goto    waitclklo           ; Jump back and wait for CLOCK low again
002D 0A47       00199      goto    timeout             ; Timed out waiting for low - check data!
002E            00200 clockok
002E 0C08       00201      movlw   .8                  ; Place initial count value into W
002F 0087       00202      subwf   COUNT,W             ; Subtract from count, place into W
0030 0743       00203      btfss   STATUS,Z            ; Skip if we are at count 8 (first value)
0031 0A34       00204      goto    skipcsumadd         ; Skip checksum add if any other count value
0032 0209       00205      movf    DATAREG,W           ; Place last byte received into W
0033 01EA       00206      addwf   CSUMTOTAL,F         ; Add to checksum
0034            00207 skipcsumadd
0034 0503       00208      bsf     STATUS,C            ; Assume data bit is high
0035 07E6       00209      btfss   PORTB,ISPDATA       ; Skip if the data bit was high
0036 0403       00210      bcf     STATUS,C            ; Set data bit to low
0037 0369       00211      rlf     DATAREG,F           ; Rotate next bit into DATAREG
0038 02E7       00212      decfsz  COUNT,F             ; Skip after 8 bits
0039 0A22       00213      goto    loopsendcal         ; Jump back and send next bit
                00214      addrtofsr ADDRPTR           ; Convert pointer address to FSR
003A 0C10           M      movlw   STARTCALBYTE        ; Place base address into W
003B 008E           M      subwf   ADDRPTR,w           ; Offset by STARTCALBYTE
003C 0024           M      movwf   FSR                 ; Place into FSR
003D 06A4           M      btfsc   FSR,5               ; Shift bits 4,5 to 5,6
003E 05C4           M      bsf     FSR,6
003F 04A4           M      bcf     FSR,5
0040 0684           M      btfsc   FSR,4
0041 05A4           M      bsf     FSR,5
0042 0584           M      bsf     FSR,4
0043 0209       00215      movf    DATAREG,W           ; Place received byte into W
0044 0020       00216      movwf   INDF                ; Move recv'd byte into CAL buffer location
0045 02AE       00217      incf    ADDRPTR,F           ; Move to the next cal byte
0046 0A20       00218      goto    loopcal             ; Go back for next byte
0047            00219 timeout
0047 0C14       00220      movlw   STARTCALBYTE+4      ; check if we received (4) params
0048 008E       00221      subwf   ADDRPTR,W           ; Move current address pointer to W
0049 0703       00222      btfss   STATUS,C            ; Skip if we have at least (4)
004A 0A93       00223      goto    sendnoise           ; not enough params - print and RESET!
004B 0200       00224      movf    INDF,W              ; Move received checksum into W
004C 00AA       00225      subwf   CSUMTOTAL,F     ; Subtract received Checksum from calc'd checksum
004D 0743       00226      btfss   STATUS,Z            ; Skip if CSUM OK
004E 0A9F       00227      goto    sendcsumbad         ; Checksum bad - print and RESET!
004F            00228 csumok
004F 0426       00229      bcf     PORTB,WORKLED       ; Turn on WORK LED
0050 0C10       00230      movlw   STARTCALBYTE        ; Place start pointer into W
0051 008E       00231      subwf   ADDRPTR,W           ; Subtract from current address
0052 002F       00232      movwf   BYTECOUNT           ; Place into number of bytes into BYTECOUNT
```

```
0053 002B      00233     movwf   TIMEHIGH           ; TEMP store into timehigh reg
0054 0C10      00234     movlw   STARTCALBYTE       ; Place start address into W
0055 002E      00235     movwf   ADDRPTR            ; Set up address pointer
0056           00236 loopprintnums
               00237     addrtofsr ADDRPTR          ; Set up FSR
0056 0C10          M     movlw   STARTCALBYTE       ; Place base address into W
0057 008E          M     subwf   ADDRPTR,w          ; Offset by STARTCALBYTE
0058 0024          M     movwf   FSR                ; Place into FSR
0059 06A4          M     btfsc   FSR,5              ; Shift bits 4,5 to 5,6
005A 05C4          M     bsf     FSR,6
005B 04A4          M     bcf     FSR,5
005C 0684          M     btfsc   FSR,4
005D 05A4          M     bsf     FSR,5
005E 0584          M     bsf     FSR,4
005F 0380      00238     swapf   INDF,W             ; Place received char into W
0060 0E0F      00239     andlw   0Fh                ; Strip off upper digits
0061 002D      00240     movwf   TEMP               ; Place into TEMP
0062 0C0A      00241     movlw   .10                ; Place .10 into W
0063 00AD      00242     subwf   TEMP,F             ; Subtract 10 from TEMP
0064 0603      00243     btfsc   STATUS,C           ; Skip if TEMP is less than 9
0065 0A6D      00244     goto    printhiletter      ; Greater than 9 - print letter instead
0066           00245 printhinumber
0066 0380      00246     swapf   INDF,W             ; Place received char into W
0067 0E0F      00247     andlw   0Fh                ; Strip off upper digits
0068 002D      00248     movwf   TEMP               ; Place into TEMP
0069 0C30      00249     movlw   '0'                ; Place ASCII '0' into W
006A 01CD      00250     addwf   TEMP,w             ; Add to TEMP, place into W
006B 09AE      00251     call    putchar            ; Send out char
006C 0A73      00252     goto    printlo            ; Jump to print next char
006D           00253 printhiletter
006D 0380      00254     swapf   INDF,W             ; Place received char into W
006E 0E0F      00255     andlw   0Fh                ; Strip off upper digits
006F 002D      00256     movwf   TEMP               ; Place into TEMP
0070 0C37      00257     movlw   'A'-.10            ; Place ASCII 'A' into W
0071 01CD      00258     addwf   TEMP,w             ; Add to TEMP, place into W
0072 09AE      00259     call    putchar            ; send out char
0073           00260 printlo
0073 0200      00261     movf    INDF,W             ; Place received char into W
0074 0E0F      00262     andlw   0Fh                ; Strip off upper digits
0075 002D      00263     movwf   TEMP               ; Place into TEMP
0076 0C0A      00264     movlw   .10                ; Place .10 into W
0077 00AD      00265     subwf   TEMP,F             ; Subtract 10 from TEMP
0078 0603      00266     btfsc   STATUS,C           ; Skip if TEMP is less than 9
0079 0A81      00267     goto    printloletter      ; Greater than 9 - print letter instead
007A           00268 printlonumber
007A 0200      00269     movf    INDF,W             ; Place received char into W
007B 0E0F      00270     andlw   0Fh                ; Strip off upper digits
007C 002D      00271     movwf   TEMP               ; Place into TEMP
007D 0C30      00272     movlw   '0'                ; Place ASCII '0' into W
007E 01CD      00273     addwf   TEMP,w             ; Add to TEMP, place into W
007F 09AE      00274     call    putchar            ; send out char
0080 0A87      00275     goto    printnext          ; jump to print next char
0081           00276 printloletter
0081 0200      00277     movf    INDF,W             ; Place received char into W
0082 0E0F      00278     andlw   0Fh                ; Strip off upper digits
0083 002D      00279     movwf   TEMP               ; Place into TEMP
0084 0C37      00280     movlw   'A'-.10            ; Place ASCII 'A' into W
0085 01CD      00281     addwf   TEMP,w             ; Add to TEMP, place into W
0086 09AE      00282     call    putchar            ; send out char
0087           00283 printnext
0087 0C7C      00284     movlw   '|'                ; Place ASCII '|' into W
0088 09AE      00285     call    putchar            ; Send out character
0089 028E      00286     incf    ADDRPTR,W          ; Go to next buffer value
008A 0E0F      00287     andlw   0Fh                ; And with F

008B 0643      00288     btfsc   STATUS,Z           ; Skip if this is NOT multiple of 16
```


© 1997 Microchip Technology Inc.

```
008C 09A9      00289     call    printcrlf          ; Print CR and LF every 16 chars
008D 02AE      00290     incf    ADDRPTR,F          ; go to next address
008E 02EF      00291     decfsz  BYTECOUNT,F        ; Skip after last byte
008F 0A56      00292     goto    loopprintnums      ; Go back and print next char
0090 09A9      00293     call    printcrlf          ; Print CR and LF
0091 05A3      00294     bsf     STATUS,PA0         ; Set page bit to page 1
Message[306]: Crossing page boundary -- ensure page bits are set.
0092 0A6B      00295     goto    programpartisp     ; Go to program part through ISP
0093           00296 sendnoise
0093 0C4E      00297     movlw   'N'                ; Place 'N' into W
0094 09AE      00298     call    putchar            ; Send char in W to terminal
0095 0C4F      00299     movlw   'O'                ; Place 'O' into W
0096 09AE      00300     call    putchar            ; Send char in W to terminal
0097 0C49      00301     movlw   'I'                ; Place 'I' into W
0098 09AE      00302     call    putchar            ; Send char in W to terminal
0099 0C53      00303     movlw   'S'                ; Place 'S' into W
009A 09AE      00304     call    putchar            ; Send char in W to terminal
009B 0C45      00305     movlw   'E'                ; Place 'E' into W
009C 09AE      00306     call    putchar            ; Send char in W to terminal
009D 09A9      00307     call    printcrlf          ; Print CR and LF
009E 0A1A      00308     goto    calget             ; RESET!
009F           00309 sendcsumbad
009F 0C43      00310     movlw   'C'                ; Place 'C' into W
00A0 09AE      00311     call    putchar            ; Send char in W to terminal
00A1 0C53      00312     movlw   'S'                ; Place 'S' into W
00A2 09AE      00313     call    putchar            ; Send char in W to terminal
00A3 0C55      00314     movlw   'U'                ; Place 'U' into W
00A4 09AE      00315     call    putchar            ; Send char in W to terminal
00A5 0C4D      00316     movlw   'M'                ; Place 'M' into W
00A6 09AE      00317     call    putchar            ; Send char in W to terminal
00A7 09A9      00318     call    printcrlf          ; Print CR and LF
00A8 0A1A      00319     goto    calget             ; RESET!
               00320
               00321 ; ****************************************
               00322 ; * printcrlf                             *
               00323 ; * Sends char .13 (Carrage Return) and   *
               00324 ; * char .10 (Line Feed) to RS-232 port   *
               00325 ; * by calling putchar.                   *
               00326 ; *      RAM used:  W                     *
               00327 ; ****************************************
00A9           00328 printcrlf
00A9 0C0D      00329     movlw   .13                ; Value for CR placed into W
00AA 09AE      00330     call    putchar            ; Send char in W to terminal
00AB 0C0A      00331     movlw   .10                ; Value for LF placed into W
00AC 09AE      00332     call    putchar            ; Send char in W to terminal
00AD 0800      00333     retlw   0                  ; Done - return!
               00334
               00335 ; ****************************************
               00336 ; * putchar                               *
               00337 ; * Print out the character stored in W   *
               00338 ; * by toggling the data to the RS-232    *
               00339 ; * output pin in software.               *
               00340 ; *     RAM used:  W,DATAREG,TEMP1       *
               00341 ; ****************************************
00AE           00342 putchar
00AE 0029      00343     movwf   DATAREG            ; Place character into DATAREG
00AF 0C09      00344     movlw   09h                ; Place total number of bits into W
00B0 0028      00345     movwf   TEMP1              ; Init TEMP1 for bit counter
00B1 0403      00346     bcf     STATUS,C           ; Set carry to send start bit
00B2 0AB4      00347     goto    putloop1           ; Send out start bit
00B3           00348 putloop
00B3 0329      00349     rrf     DATAREG,F          ; Place next bit into carry
00B4           00350 putloop1
00B4 0703      00351     btfss   STATUS,C           ; Skip if carry was set
00B5 0466      00352     bcf     PORTB,SEROUT       ; Clear RS-232 serial output bit
00B6 0603      00353     btfsc   STATUS,C           ; Skip if carry was clear
```


© 1997 Microchip Technology Inc.

```
00B7 0566       00354     bsf     PORTB,SEROUT        ; Set RS-232 serial output bit
                00355     delaybit                    ; Delay for one bit time
  0000              M     local dlylabels
                    M     ; 9600 baud, 8 bit, no parity, 104 us per bit, 52 uS per half bit
                    M     ; (8) shift/usage + (2) setup + (1) nop + (3 * 31) literal = (104) 4Mhz
00B8 0C1F           M     movlw   .31                 ; place 31 decimal literal into count
00B9 0027           M     movwf   COUNT               ; Initialize COUNT with loop count
00BA 0000           M     nop                         ; Add one cycle delay
00BB                M dlylabels
00BB 02E7           M     decfsz  COUNT,F             ; Decrement count until done
00BC 0ABB           M     goto    dlylabels           ; Not done delaying - go back!
00BD 02E8       00356     decfsz  TEMP1,F             ; Decrement bit counter, skip when done!
00BE 0AB3       00357     goto    putloop             ; Jump back and send next bit
00BF 0566       00358     bsf     PORTB,SEROUT        ; Send out stop bit
                00359     delaybit                    ; delay for stop bit
  0000              M     local dlylabels
                    M     ; 9600 baud, 8 bit, no parity, 104 us per bit, 52 uS per half bit
                    M     ; (8) shift/usage + (2) setup + (1) nop + (3 * 31) literal = (104) 4Mhz
00C0 0C1F           M     movlw   .31                 ; place 31 decimal literal into count
00C1 0027           M     movwf   COUNT               ; Initialize COUNT with loop count
00C2 0000           M     nop                         ; Add one cycle delay
00C3                M dlylabels
00C3 02E7           M     decfsz  COUNT,F             ; Decrement count until done
00C4 0AC3           M     goto    dlylabels           ; Not done delaying - go back!
00C5 0800       00360     retlw   0                   ; Done - RETURN
                00361
                00362 ; *********************************************************************
                00363 ; *  ISP routines from PICSTART-16C                                   *
                00364 ; *  Converted from PIC17C42 to PIC16C5X code by John Day             *
                00365 ; *  Originially written by Jim Pepping                              *
                00366 ; *********************************************************************
0200            00367     ORG 200                     ; ISP routines stored on page 1
                00368
                00369 ; *********************************************************************
                00370 ; * poweroffisp                                                      *
                00371 ; * Power off application PIC - turn off VPP and reset device after *
                00372 ; * programming pass is complete                                     *
                00373 ; *********************************************************************
0200            00374 poweroffisp
0200 04A6       00375     bcf     PORTB,VPPON         ; Turn off VPP 13 volts
0201 0586       00376     bsf     PORTB,GNDON         ; Apply 0 V to MCLR to reset PIC
0202 0CC1       00377     movlw   b'11000001'         ; RB6,7 set to inputs
0203 0006       00378     tris    PORTB               ; Move to tris registers
0204 0486       00379     bcf     PORTB,GNDON   ; Allow MCLR to go back to 5 volts, deassert reset
0205 0526       00380     bsf     PORTB,WORKLED       ; Turn off WORK LED
0206 0800       00381     retlw 0                     ; Done so return!
                00382
                00383 ; *********************************************************************
                00384 ; * testmodeisp                                                      *
                00385 ; * Apply VPP voltage to place application PIC into test mode.       *
                00386 ; * this enables ISP programming to proceed                          *
                00387 ; *     RAM used:       TEMP                                        *
                00388 ; *********************************************************************
0207            00389 testmodeisp
0207 0C08       00390     movlw   b'00001000'         ; Serial OFF, LEDS OFF, VPP OFF
0208 0026       00391     movwf   PORTB               ; Place "0" into port b latch register
0209 04A6       00392     bcf     PORTB,VPPON         ; Turn off VPP just in case!
020A 0586       00393     bsf     PORTB,GNDON         ; Apply 0 volts to MCLR
020B 0C01       00394     movlw   b'00000001'         ; RB6,7 set to outputs
020C 0006       00395     tris    PORTB               ; Move to tris registers
020D 0206       00396     movf    PORTB,W             ; Place PORT B state into W
020E 002D       00397     movwf   TEMP                ; Move state to TEMP
020F 048D       00398     bcf     TEMP,4              ; Turn off MCLR GND
0210 05AD       00399     bsf     TEMP,5              ; Turn on VPP voltage
0211 020D       00400     movf    TEMP,W              ; Place TEMP into W
0212 0026       00401     movwf   PORTB               ; Turn OFF GND and ON VPP
```


© 1997 Microchip Technology Inc.

```
0213 0546       00402     bsf     PORTB,DONELED       ; Turn ON GREEN LED
0214 0800       00403     retlw 0                     ; Done so return!
                00404
                00405 ; ************************************************************************
                00406 ; * p16cispout                                                           *
                00407 ; * Send 14-bit data word to application PIC for writing this data       *
                00408 ; * to it's program memory.   The data to be sent is stored in both     *
                00409 ; * HIBYTE (6 MSBs only) and LOBYTE.                                     *
                00410 ; *     RAM used:       TEMP, W, HIBYTE (inputs), LOBYTE (inputs)       *
                00411 ; ************************************************************************
0215            00412 P16cispout
0215 0C0E       00413     movlw   .14               ; Place 14 into W for bit counter
0216 002D       00414     movwf   TEMP              ; Use TEMP as bit counter
0217 04C6       00415     bcf     PORTB,ISPCLOCK    ; Clear CLOCK line
0218 04E6       00416     bcf     PORTB,ISPDATA     ; Clear DATA line
0219 0C01       00417     movlw   DATISPOUT         ; Place tris value for data output
021A 0006       00418     tris    PORTB             ; Set tris latch as data output
021B 04E6       00419     bcf     PORTB,ISPDATA     ; Send a start bit (0)
021C 05C6       00420     bsf     PORTB,ISPCLOCK    ; Set CLOCK output
021D 04C6       00421     bcf     PORTB,ISPCLOCK    ; Clear CLOCK output (clock start bit)
021E            00422 P16cispoutloop
021E 0403       00423     bcf     STATUS,C          ; Clear carry bit to start clean
021F 04E6       00424     bcf     PORTB,ISPDATA     ; Clear DATA bit to start (assume 0)
0220 0329       00425     rrf     HIBYTE,F          ; Rotate HIBYTE output
0221 032A       00426     rrf     LOBYTE,F          ; Rotate LOBYTE output
0222 0603       00427     btfsc   STATUS,C          ; Skip if data bit is zero
0223 05E6       00428     bsf     PORTB,ISPDATA     ; Set DATA line to send a one
0224 05C6       00429     bsf     PORTB,ISPCLOCK    ; Set CLOCK output
0225 04C6       00430     bcf     PORTB,ISPCLOCK    ; Clear CLOCK output (clock bit)
0226 02ED       00431     decfsz  TEMP,F            ; Decrement bit counter, skip when done
0227 0A1E       00432     goto    P16cispoutloop    ; Jump back and send next bit
0228 04E6       00433     bcf     PORTB,ISPDATA     ; Send a stop bit (0)
0229 05C6       00434     bsf     PORTB,ISPCLOCK    ; Set CLOCK output
022A 04C6       00435     bcf     PORTB,ISPCLOCK    ; Clear CLOCK output (clock stop bit)
022B 0800       00436     retlw 0                   ; Done so return!
                00437
                00438 ; ************************************************************************
                00439 ; * p16cispin                                                            *
                00440 ; * Receive 14-bit data word from application PIC for reading this       *
                00441 ; * data from it's program memory.   The data received is stored in      *
                00442 ; * both HIBYTE (6 MSBs only) and LOBYTE.                                *
                00443 ; *     RAM used:       TEMP, W, HIBYTE (output), LOBYTE (output)       *
                00444 ; ************************************************************************
022C            00445 P16cispin
022C 0C0E       00446     movlw   .14               ; Place 14 data bit count value into W
022D 002D       00447     movwf   TEMP              ; Init TEMP and use for bit counter
022E 0069       00448     clrf    HIBYTE            ; Clear recieved HIBYTE register
022F 006A       00449     clrf    LOBYTE            ; Clear recieved LOBYTE register
0230 0403       00450     bcf     STATUS,C          ; Clear carry bit to start clean
0231 04C6       00451     bcf     PORTB,ISPCLOCK    ; Clear CLOCK output
0232 04E6       00452     bcf     PORTB,ISPDATA     ; Clear DATA output
0233 0C81       00453     movlw   DATISPIN          ; Place tris value for data input into W
0234 0006       00454     tris    PORTB             ; Set up tris latch for data input
0235 05C6       00455     bsf     PORTB,ISPCLOCK    ; Send a single clock to start things going
0236 04C6       00456     bcf     PORTB,ISPCLOCK    ; Clear CLOCK to start receive
0237            00457 P16cispinloop
0237 05C6       00458     bsf     PORTB,ISPCLOCK    ; Set CLOCK bit
0238 0000       00459     nop                       ; Wait one cycle
0239 0403       00460     bcf     STATUS,C          ; Clear carry bit, assume 0 read
023A 06E6       00461     btfsc   PORTB,ISPDATA     ; Check the data, skip if it was zero
023B 0503       00462     bsf     STATUS,C          ; Set carry bit if data was one
023C 0329       00463     rrf     HIBYTE,F          ; Move recevied bit into HIBYTE
023D 032A       00464     rrf     LOBYTE,F          ; Update LOBYTE
023E 04C6       00465     bcf     PORTB,ISPCLOCK    ; Clear CLOCK line
023F 0000       00466     nop                       ; Wait one cycle
0240 0000       00467     nop                       ; Wait one cycle
```


© 1997 Microchip Technology Inc.

```
0241 02ED       00468     decfsz  TEMP,F              ; Decrement bit counter, skip when zero
0242 0A37       00469     goto    P16cispinloop       ; Jump back and receive next bit
0243 05C6       00470     bsf     PORTB,ISPCLOCK      ; Clock a stop bit (0)
0244 0000       00471     nop                         ; Wait one cycle
0245 04C6       00472     bcf     PORTB,ISPCLOCK      ; Clear CLOCK to send bit
0246 0000       00473     nop                         ; Wait one cycle
0247 0403       00474     bcf     STATUS,C            ; Clear carry bit
0248 0329       00475     rrf     HIBYTE,F            ; Update HIBYTE with the data
0249 032A       00476     rrf     LOBYTE,F            ; Update LOBYTE
024A 0403       00477     bcf     STATUS,C            ; Clear carry bit
024B 0329       00478     rrf     HIBYTE,F            ; Update HIBYTE with the data
024C 032A       00479     rrf     LOBYTE,F            ; Update LOBYTE with the data
024D 04C6       00480     bcf     PORTB,ISPCLOCK      ; Clear CLOCK line
024E 04E6       00481     bcf     PORTB,ISPDATA       ; Clear DATA line
024F 0C01       00482     movlw   DATISPOUT           ; Place tris value for data output into W
0250 0006       00483     tris    PORTB               ; Set tris to data output
0251 0800       00484     retlw 0                     ; Done so RETURN!
                00485
                00486 ; *************************************************************************
                00487 ; * commandisp                                                             *
                00488 ; * Send 6-bit ISP command to application PIC.  The command is sent *
                00489 ; * in the W register and later stored in LOBYTE for shifting.      *
                00490 ; *     RAM used: LOBYTE, W, TEMP                                         *
                00491 ; *************************************************************************
0252            00492 commandisp
0252 002A       00493     movwf   LOBYTE              ; Place command into LOBYTE
0253 0C06       00494     movlw   CMDISPCNT           ; Place number of command bits into W
0254 002D       00495     movwf   TEMP                ; Use TEMP as command bit counter
0255 04E6       00496     bcf     PORTB,ISPDATA       ; Clear DATA line
0256 04C6       00497     bcf     PORTB,ISPCLOCK      ; Clear CLOCK line
0257 0C01       00498     movlw   DATISPOUT           ; Place tris value for data output into W
0258 0006       00499     tris    PORTB               ; Set tris to data output
0259            00500 P16cispcmmdoutloop
0259 0403       00501     bcf     STATUS,C            ; Clear carry bit to start clean
025A 04E6       00502     bcf     PORTB,ISPDATA       ; Clear the DATA line to start
025B 032A       00503     rrf     LOBYTE,F            ; Update carry with next CMD bit to send
025C 0603       00504     btfsc   STATUS,C            ; Skip if bit is supposed to be 0
025D 05E6       00505     bsf     PORTB,ISPDATA       ; Command bit was a one - set DATA to one
025E 05C6       00506     bsf     PORTB,ISPCLOCK      ; Set CLOCK line to clock the data
025F 0000       00507     nop                         ; Wait one cycle
0260 04C6       00508     bcf     PORTB,ISPCLOCK      ; Clear CLOCK line to clock data
0261 02ED       00509     decfsz  TEMP,F              ; Decement bit counter TEMP, skip when done
0262 0A59       00510     goto    P16cispcmmdoutloop  ; Jump back and send next cmd bit
0263 0000       00511     nop                         ; Wait one cycle
0264 04E6       00512     bcf     PORTB,ISPDATA       ; Clear DATA line
0265 04C6       00513     bcf     PORTB,ISPCLOCK      ; Clear CLOCK line
0266 0C81       00514     movlw   DATISPIN            ; Place tris value for data input into W
0267 0006       00515     tris    PORTB               ; set as input to avoid any contention
0268 0000       00516     nop                         ; Wait one cycle
0269 0000       00517     nop                         ; Wait one cycle
026A 0800       00518     retlw 0                     ; Done - return!
                00519
                00520 ; *************************************************************************
                00521 ; * programpartisp                                                         *
                00522 ; * Main ISP programming loop.  Reads data starting at STARTCALBYTE  *
                00523 ; * and calls programming subroutines to program and verify this     *
                00524 ; * data into the application PIC.                                        *
                00525 ; *     RAM used: LOADDR, HIADDR, LODATA, HIDATA, FSR, LOBYTE, HIBYTE*
                00526 ; *************************************************************************
026B            00527 programpartisp
026B 0907       00528     call    testmodeisp         ; Place PIC into test/program mode
026C 0064       00529     clrf    FSR                 ; Point to bank 0
026D 0210       00530     movf    STARTCALBYTE,W ; Upper order address of data to be stored into W
026E 0027       00531     movwf   HIADDR              ; place into counter
026F 0211       00532     movf    STARTCALBYTE+1,W ; Lower order address byte of data to be stored
0270 0028       00533     movwf   LOADDR              ; place into counter
```


© 1997 Microchip Technology Inc.

```
0271 00E8      00534      decf    LOADDR,F          ; Subtract one from loop constant
0272 02A7      00535      incf    HIADDR,F          ; Add one for loop constant
0273           00536 programsetptr
0273 0C06      00537      movlw   CMDISPINCRADDR    ; Increment address command load into W
0274 0952      00538      call    commandisp        ; Send command to PIC
0275 02E8      00539      decfsz  LOADDR,F          ; Decrement lower address
0276 0A73      00540      goto    programsetptr     ; Go back again
0277 02E7      00541      decfsz  HIADDR,F          ; Decrement high address
0278 0A73      00542      goto    programsetptr     ; Go back again
0279 0C03      00543      movlw   .3                ; Place start pointer into W, offset address
027A 008B      00544      subwf   TIMEHIGH,W        ; Restore byte count into W
027B 002F      00545      movwf   BYTECOUNT         ; Place into byte counter
027C 0C12      00546      movlw   STARTCALBYTE+2    ; Place start of REAL DATA address into W
027D 002E      00547      movwf   ADDRPTR           ; Update pointer
027E           00548 programisploop
027E 0C34      00549      movlw   UPPER6BITS        ; retlw instruction opcode placed into W
027F 0027      00550      movwf   HIDATA            ; Set up upper bits of program word
               00551      addrtofsr ADDRPTR         ; Set up FSR to point to next value
0280 0C10          M      movlw   STARTCALBYTE      ; Place base address into W
0281 008E          M      subwf   ADDRPTR,w         ; Offset by STARTCALBYTE
0282 0024          M      movwf   FSR               ; Place into FSR
0283 06A4          M      btfsc   FSR,5             ; Shift bits 4,5 to 5,6
0284 05C4          M      bsf     FSR,6
0285 04A4          M      bcf     FSR,5
0286 0684          M      btfsc   FSR,4
0287 05A4          M      bsf     FSR,5
0288 0584          M      bsf     FSR,4
0289 0200      00552      movf    INDF,W            ; Place next cal param into W
028A 0028      00553      movwf   LODATA            ; Move it out to LODATA
028B 0208      00554      movf    LODATA,W          ; Place LODATA into LOBYTE
028C 002A      00555      movwf   LOBYTE            ;
028D 0207      00556      movf    HIDATA,W          ; Place HIDATA into HIBYTE
028E 0029      00557      movwf   HIBYTE            ;
028F 006B      00558      clrf    PULSECNT          ; Clear pulse counter
0290           00559 pgmispcntloop
0290 05E3      00560      bsf     STATUS,VFYYES     ; Set verify flag
0291 09B1      00561      call    pgmvfyisp         ; Program and verify this byte
0292 02AB      00562      incf    PULSECNT,F        ; Increment pulse counter
0293 0C19      00563      movlw   .25               ; Place 25 count into W
0294 008B      00564      subwf   PULSECNT,w        ; Subtract pulse count from 25
0295 0643      00565      btfsc   STATUS,Z          ; Skip if NOT 25 pulse counts
0296 0AA9      00566      goto    pgmispfail        ; Jump to program failed - only try 25 times
0297 0209      00567      movf    HIBYTE,w          ; Subtract programmed and read data
0298 0087      00568      subwf   HIDATA,w
0299 0743      00569      btfss   STATUS,Z          ; Skip if programmed is OK
029A 0A90      00570      goto    pgmispcntloop     ; Miscompare - program it again!
029B 020A      00571      movf    LOBYTE,w          ; Subtract programmed and read data
029C 0088      00572      subwf   LODATA,w
029D 0743      00573      btfss   STATUS,Z          ; Skip if programmed is OK
029E 0A90      00574      goto    pgmispcntloop     ; Miscompare - program it again!
029F 0040      00575      clrw                      ; Clear W reg
02A0 01CB      00576      addwf   PULSECNT,W        ; now do 3 times overprogramming pulses
02A1 01CB      00577      addwf   PULSECNT,W
02A2 01CB      00578      addwf   PULSECNT,W
02A3 002B      00579      movwf   PULSECNT          ; Add 3X pulsecount to pulsecount
02A4           00580 pgmisp3X
02A4 04E3      00581      bcf     STATUS,VFYYES     ; Clear verify flag
02A5 09B1      00582      call    pgmvfyisp         ; Program this byte
02A6 02EB      00583      decfsz  PULSECNT,F        ; Decrement pulse counter, skip when done
02A7 0AA4      00584      goto    pgmisp3X          ; Loop back and program again!
02A8 0AAA      00585      goto    prgnextbyte       ; Done - jump to program next byte!
02A9           00586 pgmispfail
02A9 0446      00587      bcf     PORTB,DONELED     ; Failure - clear green LED!
02AA           00588 prgnextbyte
02AA 0C06      00589      movlw   CMDISPINCRADDR    ; Increiment address command load into W
02AB 0952      00590      call    commandisp        ; Send command to PIC
```


© 1997 Microchip Technology Inc.

```
02AC 02AE      00591     incf    ADDRPTR,F          ; Increment pointer to next address
02AD 02EF      00592     decfsz  BYTECOUNT,F        ; See if we sent last byte
02AE 0A7E      00593     goto    programisploop     ; Jump back and send next byte
02AF 0900      00594     call    poweroffisp        ; Done - power off PIC and reset it!
02B0           00595 self
02B0 0AB0      00596     goto    self               ; Done with programming - wait here!
               00597
               00598
               00599
               00600 ; *****************************************************************
               00601 ; * pgmvfyisp                                                     *
               00602 ; * Program and/or Veryify a word in program memory on the        *
               00603 ; * application PIC.  The data to be programmed is in HIDATA and   *
               00604 ; * LODATA.                                                       *
               00605 ; *     RAM used: HIBYTE, LOBYTE, HIDATA, LODATA, TEMP           *
               00606 ; *****************************************************************
02B1           00607 pgmvfyisp
02B1           00608 loadcisp
02B1 0C02      00609     movlw   CMDISPLOAD         ; Place load data command into W
02B2 0952      00610     call    commandisp         ; Send load data command to PIC
02B3 0000      00611     nop                        ; Wait one cycle
02B4 0000      00612     nop                        ; Wait one cycle
02B5 0000      00613     nop                        ; Wait one cycle
02B6 0208      00614     movf    LODATA,w           ; Place LODATA byte into W
02B7 002A      00615     movwf   LOBYTE             ; Move it to LOBYTE reg
02B8 0207      00616     movf    HIDATA,w           ; Place HIDATA byte into W
02B9 0029      00617     movwf   HIBYTE             ; Move it to HIBYTE reg
02BA 0915      00618     call    P16cispout         ; Send data to PIC
02BB 0C08      00619     movlw   CMDISPPGMSTART     ; Place start programming command into W
02BC 0952      00620     call    commandisp         ; Send start programming command to PIC
02BD           00621 delay100us
02BD 0C20      00622     movlw   .32                ; Place 32 into W
02BE 0000      00623     nop                        ; Wait one cycle
02BF 002D      00624     movwf   TEMP               ; Move it to TEMP for delay counter
02C0           00625 loopprgm
02C0 02ED      00626     decfsz  TEMP,F             ; Decrement TEMP, skip when delay done
02C1 0AC0      00627     goto    loopprgm           ; Jump back and loop delay
02C2 0C0E      00628     movlw   CMDISPPGMEND       ; Place stop programming command into W
02C3 0952      00629     call    commandisp         ; Send end programming command to PIC
02C4 07E3      00630     btfss   STATUS,VFYYES    ; Skip if we are supposed to verify this time
02C5 0800      00631     retlw 0                    ; Done - return!
02C6 0000      00632     nop                        ; Wait one cycle
02C7           00633 readcisp
02C7 0C04      00634     movlw   CMDISPREAD         ; Place read data command into W
02C8 0952      00635     call    commandisp         ; Send read data command to PIC
02C9 092C      00636     call    P16cispin          ; Read programmed data
02CA 0800      00637     retlw 0                    ; Done - return!
               00638     END
```


© 1997 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXX---------- ---------------- ---------------- ----------------
0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0280 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
02C0 : XXXXXXXXXX----- ---------------- ---------------- ----------------
07C0 : ---------------- ---------------- ---------------- ---------------X
0FC0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   402
Program Memory Words Free:  1646


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     2 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

APPENDIX B:

```
MPASM 01.40.01 Intermediate   ISPTEST.ASM   3-31-1997  10:55:57         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ; Filename: ISPTEST.ASM
                     00002 ; ********************************************
                     00003 ; * Author:   John Day                       *
                     00004 ; *           Sr. Field Applications Engineer  *
                     00005 ; *           Microchip Technology             *
                     00006 ; * Revision: 1.0                            *
                     00007 ; * Date      August 25, 1995                *
                     00008 ; * Part:     PIC16CXX                       *
                     00009 ; * Compiled using MPASM V1.40               *
                     00010 ; ********************************************
                     00011 ; * Include files:                           *
                     00012 ; *           P16CXX.ASM                     *
                     00013 ; ********************************************
                     00014 ; * Fuses:    OSC:  XT (4.0 Mhz xtal)        *
                     00015 ; *           WDT:  OFF                      *
                     00016 ; *            CP:  OFF                      *
                     00017 ; *         PWRTE:  OFF                      *
                     00018 ; ***************************************************************************
                     00019 ; * This program is intended to be used as a code example to                *
                     00020 ; * show how to comunicate with a manufacturing test jig that               *
                     00021 ; * allows this PIC16CXX device to self program.   The RB6 and RB7           *
                     00022 ; * lines of this PIC16CXX device are used to clock the data from           *
                     00023 ; * this device to the test jig (running ISPPRGM.ASM).  Once the            *
                     00024 ; * PIC16C58 running ISPPRGM in the test jig receives the data,             *
                     00025 ; * it places this device in test mode and programs these parameters.       *
                     00026 ; * The code with comments "TEST -" is used to create some fakecalibration *
                     00027 ; * parameters that are first written to addresses STARTCALBYTE through     *
                     00028 ; * ENDCALBYTE and later used to call the self-programming algorithm.       *
                     00029 ; * Replace this code with your parameter calculation procedure,           *
                     00030 ; * placing each parameter into the STARTCALBYTE to ENDCALBYTE             *
                     00031 ; * file register addresses (16 are used in this example).  The address    *
                     00032 ; * "lookuptable" is used by the main code later on for the final lookup   *
                     00033 ; * table of calibration constants.  16 words are reserved for this lookup *
                     00034 ; * table.                                                                 *
                     00035 ; ***************************************************************************
                     00036 ; * Program Memory:                                                        *
                     00037 ; *             49 Words - communication with test jig                    *
                     00038 ; *             17 Words - calibration look-up table (16 bytes of data)    *
                     00039 ; *             13 Words - Test Code to generate Calibration Constants     *
                     00040 ; * RAM Memory:                                                            *
                     00041 ; *             16 Bytes -Temporary- Store 16 bytes of calibration constant*
                     00042 ; *              4 Bytes -Temporary- Store 4 bytes of temp variables       *
                     00043 ; ***************************************************************************
                     00044
Warning[217]: Hex file format specified on command line.
                     00045     list p=16C71,f=inhx8m
                     00046     include <p16C71.inc>
                     00001         LIST
                     00002 ; P16C71.INC  Standard Header File, Version 1.00 Microchip Technology, Inc.
                     00142         LIST
2007 3FF1            00047     __CONFIG _CP_OFF&_WDT_OFF&_XT_OSC&_PWRTE_OFF
                     00048
                     00049 ; ***********************************
                     00050 ; * Port A (RA0-RA4) bit definitions *
                     00051 ; ***********************************
                     00052 ; Port A is not used in this test program
                     00053
                     00054 ; ***********************************
                     00055 ; * Port B (RB0-RB7) bit definitions *
```


© 1997 Microchip Technology Inc.

```
                        00056 ; ************************************
                        00057 #define     CLOCK   6 ; clock line for ISP
                        00058 #define     DATA    7 ; data line for ISP
                        00059 ; Port pins RB0-5 are not used in this test program
                        00060
                        00061 ; ************************************
                        00062 ; * RAM register usage definition    *
                        00063 ; ************************************
  0000000C              00064 CSUMTOTAL   EQU 0Ch ; Address for checksum var
  0000000D              00065 COUNT       EQU 0Dh ; Address for COUNT var
  0000000E              00066 DATAREG     EQU 0Eh ; Address for Data output register var
  0000000F              00067 COUNTDLY    EQU 0Fh ; Address for clock delay counter
                        00068
                        00069 ; These two symbols are used for the start and end address
                        00070 ; in RAM where the calibration bytes are stored.  There are 16 bytes
                        00071 ; to be stored in this example; however, you can increase or
                        00072 ; decrease the number of bytes by changing the STARTCALBYTE or ENDCALBYTE
                        00073 ; address values.
                        00074
  00000010              00075 STARTCALBYTE    EQU 10h     ; Address pointer for start CAL byte
  0000002F              00076 ENDCALBYTE      EQU 2Fh     ; Address pointer for end CAL byte
                        00077
                        00078 ; Table length of lookup table (number of CAL parameters to be stored)
                        00079
  00000020              00080 CALTABLELENGTH EQU  ENDCALBYTE - STARTCALBYTE + 1
                        00081
0000                    00082     ORG 0
                        00083 ; ******************************************************************
                        00084 ; * testcode routine                                              *
                        00085 ; * TEST code - sets up RAM register with register address as data *
                        00086 ; * Uses file register STARTCALBYTE through ENDCALBYTE to store the*
                        00087 ; * calibration values that are to be programmed into the lookup    *
                        00088 ; * table by the test jig running ISPPRGM.                         *
                        00089 ; * Customer would place calibration code here and make sure that  *
                        00090 ; * calibration constants start at address STARTCALBYTE           *
                        00091 ; ******************************************************************
0000                    00092 testcode
0000 3010               00093     movlw   STARTCALBYTE    ; TEST -
0001 0084               00094     movwf   FSR             ; TEST - Init FSR with start of RAM addres
0002                    00095 looptestram
0002 0804               00096     movf    FSR,W           ; TEST - Place address into W
0003 0080               00097     movwf   INDF            ; TEST - Place address into RAM data byte
0004 0A84               00098     incf    FSR,F           ; TEST - Move to next address
0005 0804               00099     movf    FSR,W           ; TEST - Place current address into W
0006 3C30               00100     sublw   ENDCALBYTE+1    ; TEST - Subtract from end of RAM
0007 1D03               00101     btfss   STATUS,Z        ; TEST - Skip if at END of ram
0008 2802               00102     goto    looptestram     ; TEST - Jump back and init next RAM byte
0009 0103               00103     clrw                    ; TEST - Clear W
000A 200F               00104     call    lookuptable     ; TEST - Get first CAL value from lookup table
000B 3CFF               00105     sublw   0FFh            ; TEST - Check if lookup CAL table is blank
000C 1903               00106     btfsc   STATUS,Z        ; TEST - Skip if table is NOT blank
000D 2830               00107     goto    calsend         ; TEST - Table blank - send out cal parameters
000E                    00108 mainloop
000E 280E               00109     goto    mainloop        ; TEST - Jump back to self since CAL is done
                        00110
                        00111 ; ******************************************************************
                        00112 ; * lookuptable                                                   *
                        00113 ; * Calibration constants look-up table.  This is where the CAL    *
                        00114 ; * Constants will be stored via ISP protocol, later.  Note it is  *
                        00115 ; * blank, since these values will be pogrammed by the test jig    *
                        00116 ; * running ISPPRGM later.                                        *
                        00117 ; *     Input Variable:  W stores index for table lookup          *
                        00118 ; *     Output Variable: W returns with the calibration constant  *
                        00119 ; * NOTE: Blank table when programmed reads "FF" for all locations *
                        00120 ; ******************************************************************
000F                    00121 lookuptable
```

```
000F 0782       00122     addwf   PCL,F           ; Place the calibration constant table here!
                00123
002F            00124     ORG     lookuptable + CALTABLELENGTH
002F 34FF       00125     retlw   0FFh            ; Return FF at last location for a blank table
                00126
                00127 ; ********************************************************************
                00128 ; * calsend subroutine                                               *
                00129 ; * Send the calibration data stored in locations STARTCALBYTE      *
                00130 ; * through ENDCALBYTE in RAM to the programming jig using a serial*
                00131 ; * clock and data protocol                                         *
                00132 ; *     Input Variables:   STARTCALBYTE through ENDCALBYTE         *
                00133 ; ********************************************************************
0030            00134 calsend
0030 018C       00135     clrf    CSUMTOTAL       ; Clear CSUMTOTAL reg for delay counter
0031 018D       00136     clrf    COUNT           ; Clear COUNT reg to delay counter
0032            00137 delayloop                   ; Delay for 100 mS to wait for prog jig wakeup
0032 0B8D       00138     decfsz  COUNT,F         ; Decrement COUNT and skip when zero
0033 2832       00139     goto    delayloop       ; Go back and delay again
0034 0B8C       00140     decfsz  CSUMTOTAL,F     ; Decrement CSUMTOTAL and skip when zero
0035 2832       00141     goto    delayloop       ; Go back and delay again
0036 0186       00142     clrf    PORTB           ; Place "0" into port b latch register
0037 1683       00143     bsf     STATUS,RP0      ; Switch to bank 1
0038 303F       00144     movlw   b'00111111'     ; RB6,7 set to outputs
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0039 0086       00145     movwf   TRISB           ; Move to TRIS registers
003A 1283       00146     bcf     STATUS,RP0      ; Switch to bank 0
003B 018C       00147     clrf    CSUMTOTAL       ; Clear checksum total byte
003C 3001       00148     movlw   high lookuptable+1 ; place MSB of first addr of cal table into W
003D 204D       00149     call    sendcalbyte     ; Send the high address out
003E 3010       00150     movlw   low lookuptable+1  ; place LSB of first addr of cal table into W
003F 204D       00151     call    sendcalbyte     ; Send low address out
0040 3010       00152     movlw   STARTCALBYTE    ; Place RAM start address of first cal byte
0041 0084       00153     movwf   FSR             ; Place this into FSR
0042            00154 loopcal
0042 0800       00155     movf    INDF,W          ; Place data into W
0043 204D       00156     call    sendcalbyte     ; Send the byte out
0044 0A84       00157     incf    FSR,F           ; Move to the next cal byte
0045 0804       00158     movf    FSR,W           ; Place byte address into W
0046 3C30       00159     sublw   ENDCALBYTE+1    ; Set Z bit if we are at the end of CAL data
0047 1D03       00160     btfss   STATUS,Z        ; Skip if we are done
0048 2842       00161     goto    loopcal         ; Go back for next byte
0049 080C       00162     movf    CSUMTOTAL,W     ; place checksum total into W
004A 204D       00163     call    sendcalbyte     ; Send the checksum out
004B 0186       00164     clrf    PORTB           ; clear out port pins
004C            00165 calsenddone
004C 284C       00166     goto    calsenddone     ; We are done - go home!
                00167
                00168 ; ********************************************************************
                00169 ; * sendcalbyte subroutine                                          *
                00170 ; * Send one byte of calibration data to the programming jig       *
                00171 ; *     Input Variable: W contains the byte to be sent             *
                00172 ; ********************************************************************
004D            00173 sendcalbyte
004D 008E       00174     movwf   DATAREG         ; Place send byte into data register
004E 078C       00175     addwf   CSUMTOTAL,F     ; Update checksum total
004F 3008       00176     movlw   .8              ; Place 8 into W
0050 008D       00177     movwf   COUNT           ; set up counter register
0051            00178 loopsendcal
0051 1706       00179     bsf     PORTB,CLOCK     ; Set clock line high
0052 205C       00180     call    delaysend       ; Wait for test jig to synch up
0053 0D8E       00181     rlf     DATAREG,F       ; Rotate to next bit
0054 1786       00182     bsf     PORTB,DATA      ; Assume data bit is high
0055 1C03       00183     btfss   STATUS,C        ; Skip if the data bit was high
0056 1386       00184     bcf     PORTB,DATA      ; Set data bit to low
0057 1306       00185     bcf     PORTB,CLOCK     ; Clear clock bit to clock data out
0058 205C       00186     call    delaysend       ; Wait for test jig to synch up
```


© 1997 Microchip Technology Inc.

```
0059 0B8D      00187     decfsz  COUNT,F         ; Skip after 8 bits
005A 2851      00188     goto    loopsendcal     ; Jump back and send next bit
005B 0008      00189     return                  ; We are done with this byte so return!
               00190
               00191 ; ********************************************************************
               00192 ; * delaysend subroutine                                             *
               00193 ; * Delay for 50 ms to wait for the programming jig to synch up      *
               00194 ; ********************************************************************
005C           00195 delaysend
005C 3010      00196     movlw   10h             ; Delay for 16 loops
005D 008F      00197     movwf   COUNTDLY        ; Use COUNTDLY as delay count variable
005E           00198 loopdelaysend
005E 0B8F      00199     decfsz  COUNTDLY,F      ; Decrement COUNTDLY and skip when done
005F 285E      00200     goto    loopdelaysend   ; Jump back for more delay
0060 0008      00201     return
               00202         END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX ---------------- ---------------X XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX X--------------- ----------------
2000 : -------X-------- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:    66
Program Memory Words Free:   958


Errors   :     0
Warnings :     1 reported,     0 suppressed
Messages :     1 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN658

LCD Fundamentals Using PIC16C92X Microcontrollers

Author: *Rodger Richey*
Microchip Technology Inc.

INTRODUCTION

This Application Note provides a basic introduction to the features and uses of Liquid Crystal Displays (LCD).

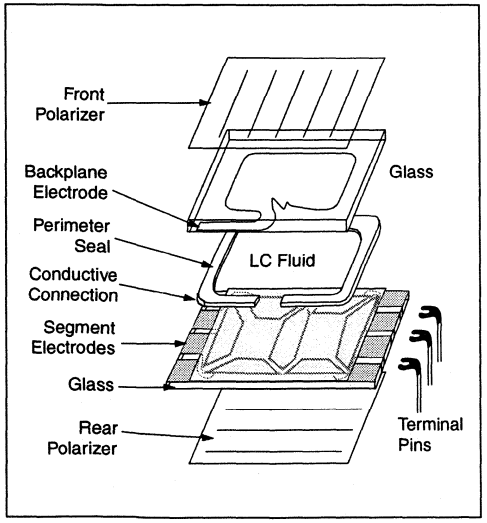
At the end of this Application Note, you should be able to answer the following questions:

- What are the basic components in an LCD panel?
- How does an LCD work?
- What are the different types of LCD panels?
- How are LCD panels driven?

WHAT ARE THE BASIC COMPONENTS IN AN LCD PANEL?

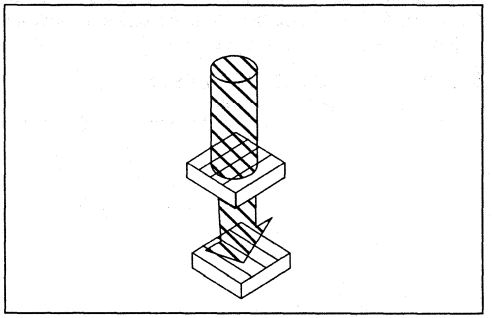
An LCD panel, or more commonly known as a piece of "glass", is constructed of many layers. Figure 1 shows all the layers that are typically present in LCD panels. The first layer is called the **front polarizer**.

FIGURE 1: BASIC LCD COMPONENTS



Polarization is a process or state in which rays of light exhibit different properties in different directions, especially the state in which all the vibration takes place in one plane. Essentially, a polarizer passes light only in one plane. As shown in Figure 2, if light is polarized in one plane, by passing through a polarizer, it cannot pass through a second polarizer if its plane is 90° out of phase to the first.

FIGURE 2: POLARIZERS OUT OF PHASE



The front polarizer is applied to the outside surface of the top piece of **glass**. The top piece of glass also provides structural support for the LCD panel.

On the bottom of the top glass, a transparent coating of Indium-Tin Oxide (ITO) is applied to the glass. ITO is conductive and forms the **backplane** or common electrodes of the LCD panel. The patterns of the backplane and segment ITO forms the numbers, letters, symbols, icons, etc.

After the ITO has been applied to the glass, a thin polyimide coating is applied to the ITO. The polyimide is "rubbed" in a single direction that matches the polarization plane of the front polarizer. The action of "rubbing" the polyimide causes the Liquid Crystal (LC) molecules in the outermost plane to align themselves in the same direction.

3

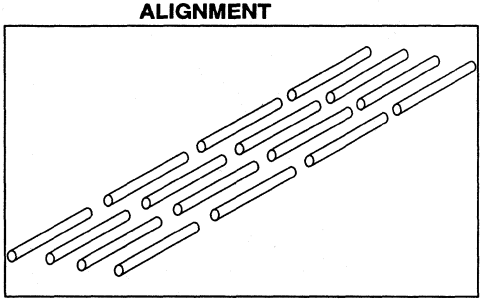
© 1997 Microchip Technology Inc.

The next layer is a reservoir of LC. The **LC fluid** has many planes of molecules.

The next layer is the polyimide coating on the bottom glass followed by the ITO **segment electrodes**. The bottom glass also supplies structural integrity for the LCD panel as well as mounting surface for the electrode connections. Applied to the external surface of the bottom glass is the rear polarizer. Depending on the type of viewing mode employed by the LCD panel, the axis of polarization is the same or 90° apart from the front polarizer.

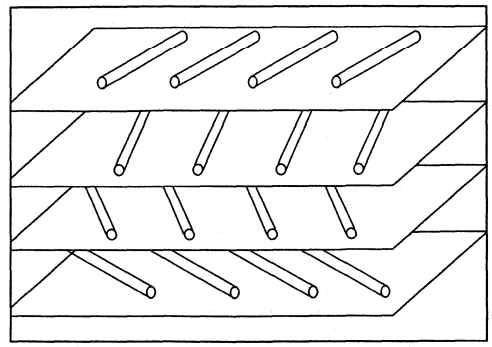
LC molecules are long and cylindrical. On any plane within the LC fluid, the molecules align themselves such that the major axis of each molecule is parallel to all others, as shown in Figure 3. The outermost planes of LC molecules will align themselves on the same axis that the polyimide is "rubbed". The direction of "rubbing" of the polyimide on the bottom glass is 90° apart from that of the polyimide on the top glass. This orientation creates the twist in the LC fluid.

FIGURE 3: LC MOLECULES IN ALIGNMENT



A consequence of this alignment is that each intermediate plane of LC molecules will have a slightly different orientation from the plane above or below as seen in Figure 4.

FIGURE 4: LC MOLECULES PLANE ORIENTATION



The twisting of the planes causes the polarization of the light to twist as it passes through the LC fluid. The twisting of the LC planes is critical to the operation of the LCD panel as will be shown in the next section.

Now that the mystery of what the LCD panel is made of has been uncovered, how does an LCD work?

© 1997 Microchip Technology Inc.

HOW DOES AN LCD WORK?

As explained before, the twist created in the LC fluid is the basis of how the panel operates. Figure 5 shows how an LCD panel creates a pixel that is OFF. For this example the LC fluid is not energized, i.e. there is 0 VRMS potential between the backplane and segment electrodes. The following is a step-by-step description of the path light takes through the LCD panel.

1. Light enters the panel through the rear polarizer. At this point the light becomes polarized to the vertical plane.
2. The polarized light passes unobstructed through the transparent backplane electrode.
3. As the polarized light passes through the LC fluid it gets twisted into the horizontal plane.
4. The polarized light passes unobstructed through the transparent segment electrode.
5. Since the light is now polarized in the horizontal plane, it passes unobstructed through the front polarizer which has a horizontal polarization.
6. The observer does not detect that the pixel is on because the light has not been obstructed.

If a potential is applied across the backplane and segment electrodes, the LC fluid becomes energized. The LC molecule planes will now align themselves such that they are parallel to the electrical field generated by the potential difference. This removes the twisting effect of the LC fluid. Figure 6 shows a pixel that is ON or, more specifically energized. The following is a step-by-step description of the path that the light takes through this LCD panel.

1. Light enters the panel through the rear polarizer. At this point the light becomes polarized to the vertical plane.
2. The polarized light passes unobstructed through the transparent backplane electrode.
3. As the polarized light passes through the LC fluid it does not twist and remains in the vertical plane.
4. The polarized light passes unobstructed through the transparent segment electrode.
5. Since the light is still polarized in the vertical plane, it is obstructed by the front polarizer which has a horizontal polarization.
6. The observer detects that the pixel is on because the light has been obstructed and creates a dark image on the panel.

FIGURE 5: PATH OF LIGHT FOR OFF PIXEL (POSITIVE IMAGE)

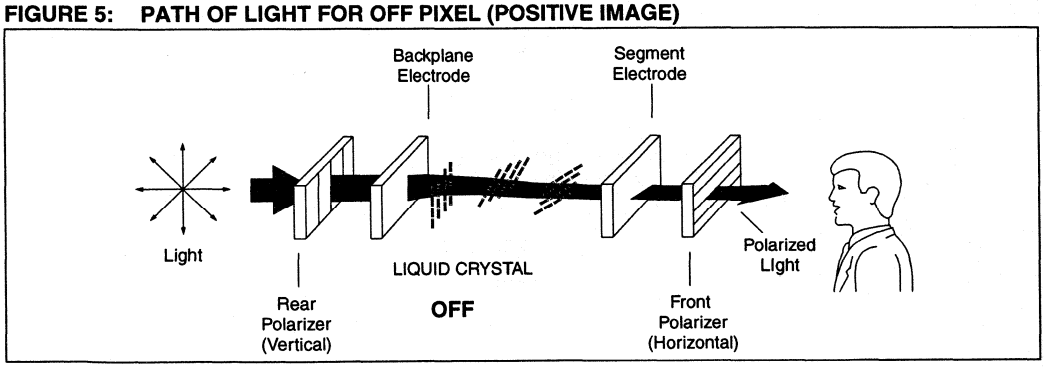
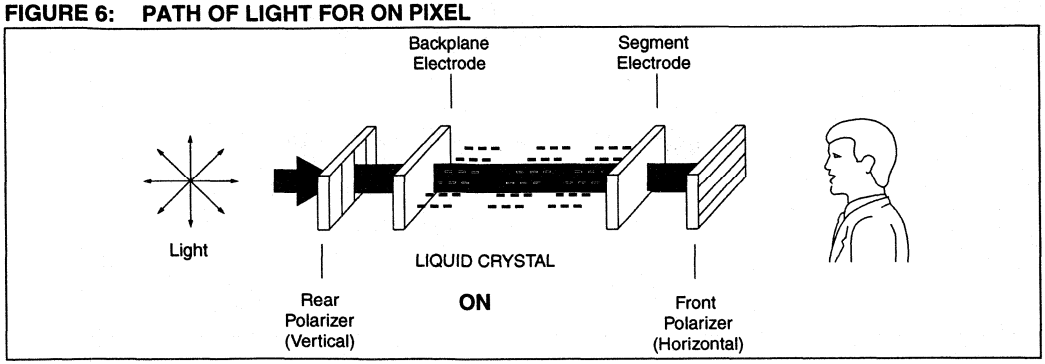


FIGURE 6: PATH OF LIGHT FOR ON PIXEL



© 1997 Microchip Technology Inc.

LCD IMAGES

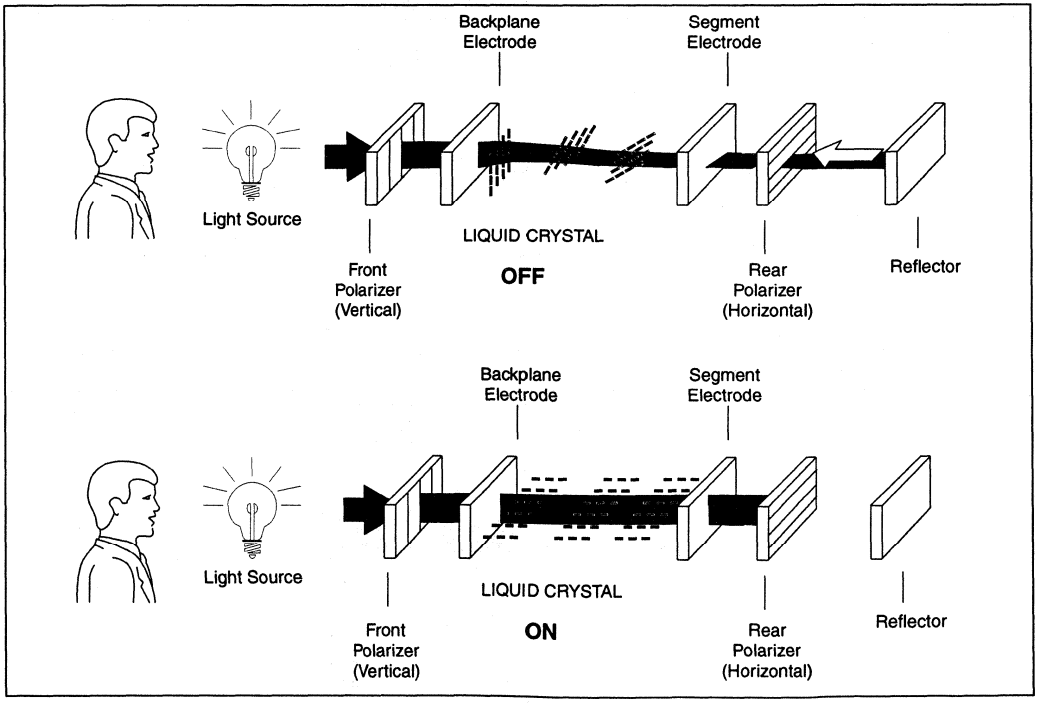
LCDs have the capability to produce both positive and negative images. A **positive image** is defined to be a dark image on a light background. In a positive image display, the front and rear polarizers are perpendicular to each other. Unenergized pixels and the background area transmit the light and energized pixels obstruct the light creating dark images on the light background. A **negative image** is a light image on a dark background. In this type of display, the front and rear polarizers are aligned to each other. Unenergized pixels and the background inhibit light from passing through the display. Energized pixels allow the light to pass creating a light image on a dark background.

There are essentially three types of viewing modes for a LCD: reflective, transmissive, and transflective. Typically **Reflective displays** use only positive images. The front and rear polarizers are perpendicular to each other. The LCD panel will have an additional layer added to the bottom of the display, a reflector. Figure 7 shows the diagrams for pixels that are ON and OFF for reflective displays. Here again, the path that light takes is described in a step-by-step fashion for a pixel that is OFF in a positive image display.

1. Light enters the panel through the front polarizer. At this point the light becomes polarized to the vertical plane.
2. The polarized light passes unobstructed through the transparent backplane electrode.
3. As the polarized light passes through the LC fluid it gets twisted into the horizontal plane.
4. The polarized light passes unobstructed through the transparent segment electrode.
5. Since the light is now polarized in the horizontal plane, it passes unobstructed through the rear polarizer which has a horizontal polarization.
6. The reflector behind the rear polarizer reflects the incoming light back on the same path.
7. The observer does not detect that the pixel is ON because the light was reflected back.

A pixel that is ON follows the same basic steps except that the light never reaches the reflector and therefore does not return to the observer. Reflective displays lend themselves to battery powered applications because the images are created using ambient light sources. These displays are very bright under proper lighting conditions, with excellent contrast, and have a wide viewing angle.

FIGURE 7: REFLECTIVE LCD PATH OF LIGHT



© 1997 Microchip Technology Inc.

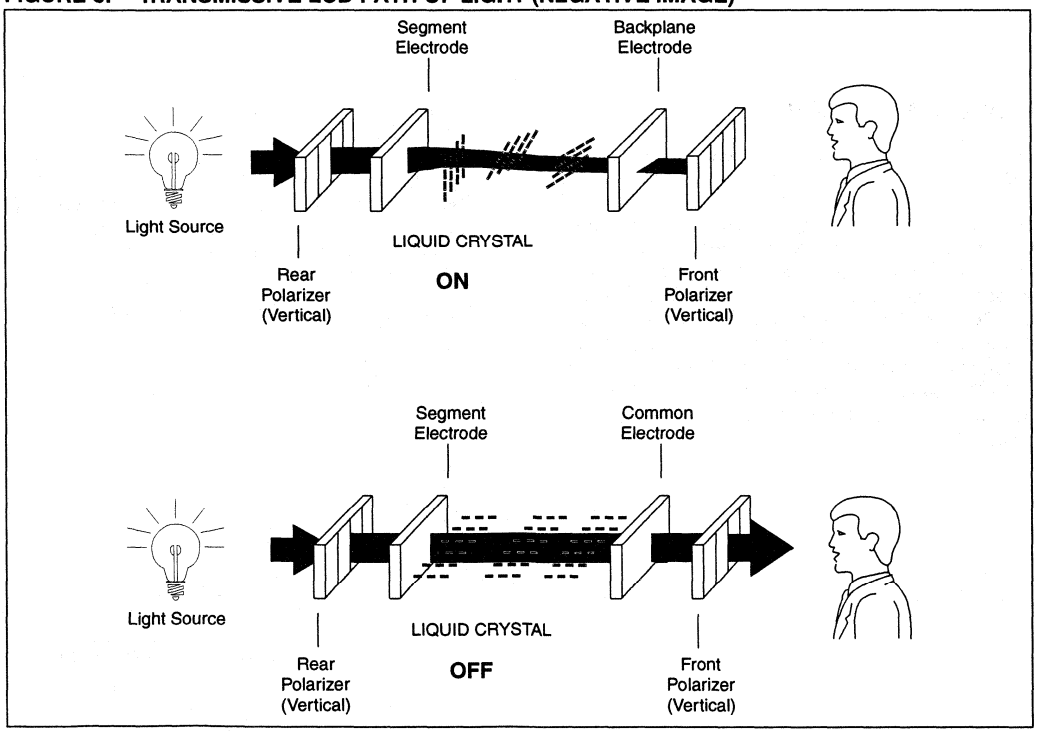
Transmissive displays do not reflect light back to the observer. Instead, they rely upon a light source behind the panel to create images. A transmissive display has front and rear polarizers that are in phase to each other. Figure 8 shows the OFF and ON diagrams for a transmissive display. The path of light is described below for the ON state only in a positive image display.

1. Light enters the panel through the rear polarizer. At this point the light becomes polarized to the vertical plane.
2. The polarized light passes unobstructed through the transparent segment electrode.
3. As the polarized light passes through the LC fluid it gets twisted into the horizontal plane.
4. The polarized light passes unobstructed through the transparent backplane electrode.
5. Since the light is now polarized in the horizontal plane, it is obstructed by the front polarizer which has a vertical polarization. Very little light passes through the front polarizer
6. The observer does not detect that the pixel is ON because the light was obstructed.

An OFF pixel would allow the light to pass through the display unobstructed because the polarization does not get twisted by the LC fluid. These displays are very good for very low light level conditions. They are very poor when used in direct sunlight because the sunlight swamps out the backlighting.

The third type of display is called **transflective**. As you can probably tell from the name, it is a combination of reflective and transmissive. A white or silver translucent material is applied to the rear of the display. It reflects some of the ambient light back to the observer while also allowing backlighting. Transflective displays are very good for applications which have varying light conditions such as gas pumps. They must operate during the day in bright sunlight, but must also operate at night. Transflective displays have lower contrast ratios than reflective displays because some of the light passes through the reflector.

FIGURE 8: TRANSMISSIVE LCD PATH OF LIGHT (NEGATIVE IMAGE)



© 1997 Microchip Technology Inc.

The type of LCD that an application requires is largely dependent on the ambient light available. Table 1 gives some guidelines for selecting a display according to the lighting conditions.

TABLE 1: LIGHTING CONDITION REFERENCE

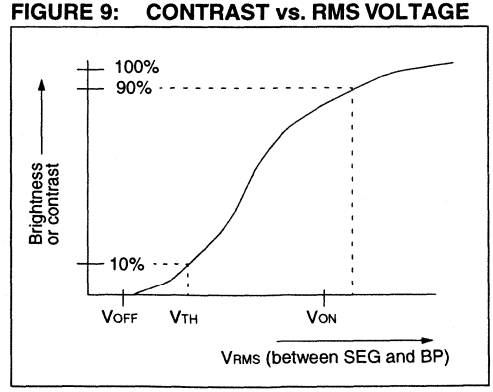
Viewing Mode	Display Description	Application Comments	Direct Sunlight	Office Light	Very Low Light
Reflective (Positive)	Dark images on light background	No backlighting. Gives best contrast and environmental stability	Excellent	Very Good	Unusable
Transflective (Positive)	Dark images on gray background	Can be viewed with both ambient light and backlighting	Excellent (no backlight)	Good (no backlight)	Very Good (backlight)
Transflective (Negative)	Light gray images on dark background	Requires high ambient light or backlighting.	Good (no backlight)	Fair (no backlight)	Very Good (backlight)
Transmissive (Negative)	Backlit images on dark background	Cannot be viewed by reflection	Poor (backlight)	Good (backlight)	Excellent (backlight)
Transmissive (Positive)	Dark images on a backlit background	Good for very low light conditions	Poor (backlight)	Good (backlight)	Excellent (backlight)

DRIVER VOLTAGES

The number one cause of LCD damage is having a DC voltage applied to it. A DC voltage will deteriorate the LC fluid such that it cannot be energized. The LCD driver waveforms are designed to create a 0 VDC potential across all pixels. The specifications for a LCD panel will include some RMS voltages such as **VOFF** and **VON**. A third voltage is **VTH** which is the RMS voltage across an LCD pixel when contrast reaches a 10% level. Often this voltage is used as VOFF. VON is defined as the RMS voltage applied by the LCD driver to the segment electrode that creates an ON pixel which is typically at the 90% contrast level. It is desirable that VON be much greater than VOFF.

Figure 9 graphically represents the voltage potential versus the contrast across a pixel. The final specification for an LCD panel is the discrimination ratio which is VON divided by VOFF (VON/VOFF). The **discrimination ratio** specifies what type of contrast levels the LCD panel will be able to achieve. Examples of discrimination ratio calculations will be given in the section "How are LCD Panels Driven?".

FIGURE 9: CONTRAST vs. RMS VOLTAGE

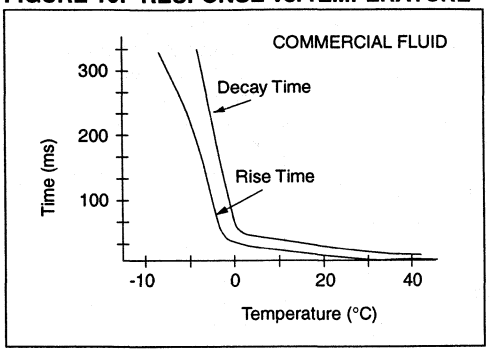


© 1997 Microchip Technology Inc.

RESPONSE TIME

An LCD panel will have a typical ON and OFF response time. The **ON time** parameter refers to the time for an OFF pixel to become visible after the appropriate voltages have been applied. The **OFF time** parameter specifies the time for an ON segment to disappear. Sometimes these parameters are called rise and decay, respectively. Temperature plays a key role in the response time of an LCD panel. Figure 10 shows the response times versus temperature for commercial type LC fluid. For this reason, there are no LCD panels in gas pumps in Alaska without heaters. Displays with heaters can help to decrease response time even at temperatures as low as -55°C. The drawback of an LCD heater is that every square inch of surface on the back of the display requires 2 to 3 watts.

FIGURE 10: RESPONSE vs. TEMPERATURE



TEMPERATURE EFFECTS

As previously shown, temperature has a large impact on the performance of the LCD panel. Not only is the LC fluid affected, but the internal coatings begin to deteriorate. All LC fluids have well defined operating temperature limits. If an LCD is operated above its fluid limits, the LC molecules begin to assume random orientations. The pixels on a positive image display will become completely dark, while pixels on a negative image display will become completely transparent. An LCD can recover from these conditions if the exposure is kept short, however, temperatures above 110°C will cause the ITO and polyimide coatings to deteriorate.

On the low end of the temperature spectrum, response times increase because the viscosity of the LC fluid increases. At very low temperatures, typically -60°C, the LC fluid transitions into a crystalline state. Usually, the LC fluid can recover from the effects of low temperature. Many different types of LC fluid are available, which allows the LCD panel to be tailored to the expected operating conditions. As mentioned in the previous section, heaters can combat the effects of low temperature.

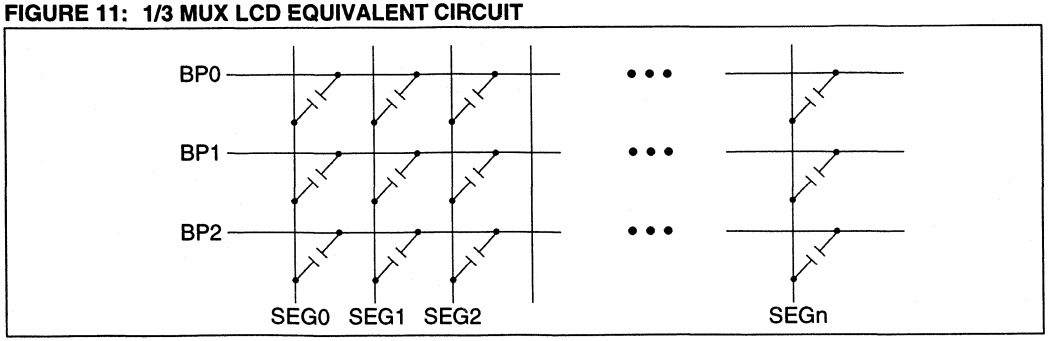
CAPACITANCE

The LCD panel can be modeled as a lossy, non-linear capacitor. The area of the pixel, and therefore the size of the LCD panel, has a direct impact on the value of the capacitance that a common or segment driver must be able to drive. Typical values of capacitance are in the range of 1000 - 1500 pF/cm^2. Figure 11 shows an example of a 1/3 MUX panel. As you can see the backplane driver must be capable of driving significantly higher capacitances than the segment driver.

Care must be taken when designing a system such that your LCD driver is capable of driving the capacitance on the segment and common. Otherwise the LCD panel may be damaged due to a DC offset voltage generated by overloaded segment and common drivers.

PIC16C92X microcontrollers are capable of driving backplanes up to 5000 pF and segments up to 500 pF.

FIGURE 11: 1/3 MUX LCD EQUIVALENT CIRCUIT



© 1997 Microchip Technology Inc.

BACKLIGHTING

A variety of methods exist for backlighting LCD panels, such as, incandescent lamps, LEDs, and electroluminescent lamps. Incandescent lamps require some type of reflector to provide uniform lighting to all areas of the panel. LEDs require some type of lightguide or lightpipe to evenly distribute light. Electroluminescent lamps typically come in some type of a panel arrangement. Other lighting methods are available for specific applications, such as fluorescent. Table 2 provides a comparison of these methods of backlighting.

TABLE 2: BACKLIGHTING FEATURES COMPARISON

Feature	LED	Incandescent	Electroluminescent
Brightness	Medium	High	Low-Medium
Color	Limited	White	White
Size	Small	Small-Medium	Thin
Voltage	5V	1.5V - 28V	45V - 100V
Current @5V/sq. in	10-30 mA	20 mA	1 mA - 10 mA
Temperature	Warm	Hot	Cool
Cost/sq. in	$0.10 - $1.00	$0.10 - $0.80	$0.50 - $2.00
Shock Tolerance	Excellent	Fragile	Excellent
Life (hours)	100,000	150 - 10,000	500 - 15,000

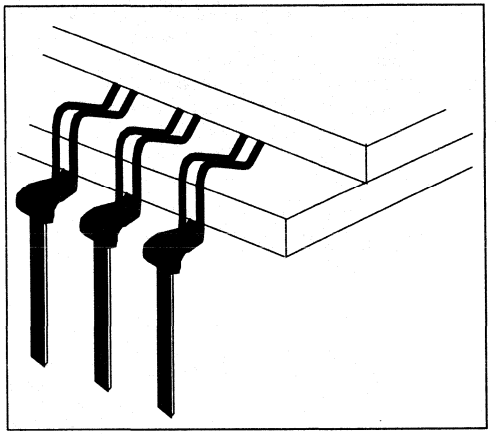
© 1997 Microchip Technology Inc.

CONNECTION METHODS

The first method of connecting the LCD panel to the world was the dual-in-line pin shown in Figure 12.

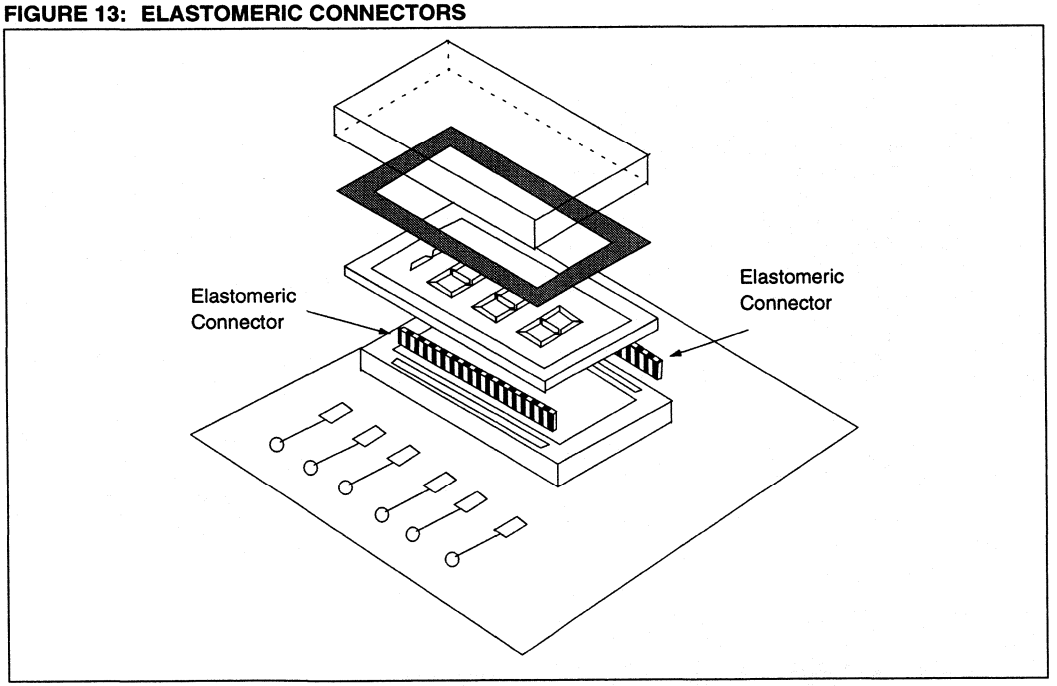
These pins provide excellent protection for harsh environments, vibration or shock. The LCD panel is either soldered directly to the printed circuit board (PCB) or inserted into headers.

FIGURE 12: DUAL IN-LINE PINS



The second method is elastomeric connectors. This method allows fast assembly/disassembly without having to solder the LCD panel. Elastomeric connectors are used on small applications where space is a concern. These connectors are relatively resistant to shock and vibration, but special consideration must be used when the panel will be exposed to harsh environments. Figure 13 shows an assembly drawing of an elastomeric connector.

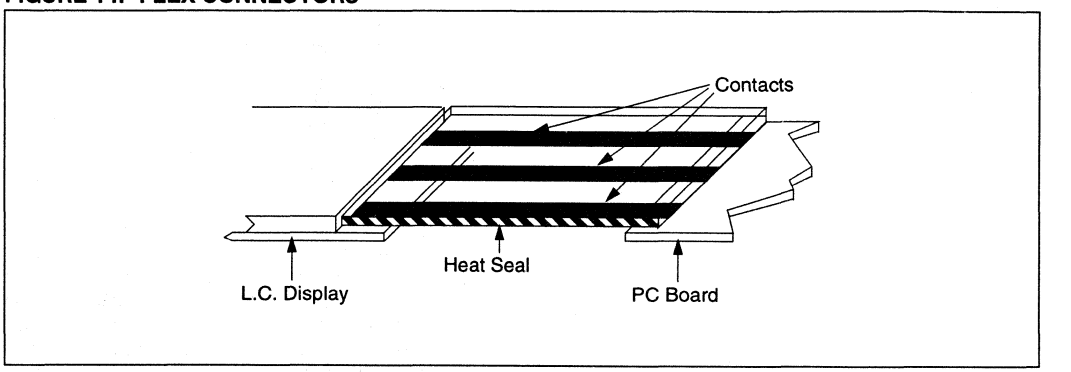
FIGURE 13: ELASTOMERIC CONNECTORS



© 1997 Microchip Technology Inc.

One of the newer methods is the flex connector. A PCB and the LCD panel are connected by a flexible cable using a heat seal process. The flexible cable is typically a anisotropic connective film that is applied to the PCB and LCD panel using heat and pressure. These connectors were designed for harsh environments where the connector must be flexible enough to prevent breakage during stress. These connectors are becoming more popular with large or remotely mounted LCD panels. Figure 14 shows a typical application.

FIGURE 14: FLEX CONNECTORS



WHAT ARE THE DIFFERENT TYPES OF LCD PANELS?

LCD panels come in many flavors depending on the application and the operating environment. LCDs can be classified in two ways. First of all, LCDs come in direct drive or multiplex drive variations. Direct drive, otherwise known as static, means that each pixel of the LCD panel has an independent driver. The LCD panel also has only one backplane. A static drive panel also has static bias. Bias is defined as the number of voltage levels the LCD driver uses to create images on the screen. The number of voltage levels is equivalent to the 1 + 1/bias. Static bias refers to two voltage levels which create a square wave, ground and V_{DD}. Static drive panels also have the best contrast ratios over the widest temperature range.

Multiplex drive panels reduce the overall amount of interconnections between the LCD and the driver. Put simply, multiplex panels have more than one backplane. A multiplex LCD driver produces an amplitude-varying, time synchronized waveform for both the segment and backplanes. These waveforms allow access to one pixel on each of the backplanes. This significantly increases the complexity of the driver. The number of backplanes a panel has is referred to the multiplexing ratio or "MUX" of the panel. MUX also refers to duty cycle. For instance, a 1/3 MUX panel has three backplanes. The bias for multiplex panels is at least 1/2 - 1/5 for segment type drivers and from 1/8 - 1/33 for dot matrix. Table 3 illustrates the advantage of multiplex panels.

TABLE 3: STATIC vs. MULTIPLEX PIN COUNT

LCD panel	Back planes	Segments	Total
3 - 1/2 digit	1	23	24
	2	12	14
8 digits	1	64	65
	4	16	20
2 x 16 character dot matrix, 5 x 7 characters	1	1280	1281
	8	160	168
	16	80	96
128 x 240 graphic display	1	30,720	30721
	64	480	544
	128	240	368

The last time Microchip investigated high pin count packages, 30,000+ was not an option.

© 1997 Microchip Technology Inc.

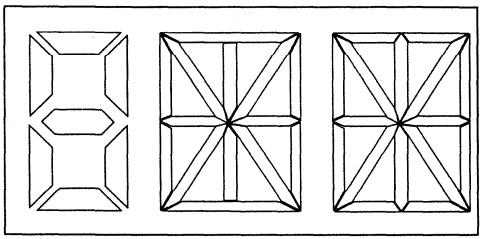
PIC16C92X microcontrollers have the following drive capabilities:

TABLE 4: PIC16C92X DRIVE CAPABILITY

MUX	Bias	Backplanes	Segments
Static	Static	1	32
1/2	1/3	2	31
1/3	1/3	3	30
1/4	1/3	4	29

The other method of classifying LCD panels is the type of display notation used, i.e. segment, dot matrix, or functional. Segment displays are usually the 7-segment, 14-segment, or 16-segment ("British Flag") types used to create numbers and letters. These type of displays are static drive which provides the best contrast and readability in sunlight. Figure 15 shows all three segment displays mentioned.

FIGURE 15: SEGMENT TYPE DISPLAY



Dot matrix displays are always multiplex type displays due to the large number of pixels required and pin limitations on the driver. Dot matrix displays can create more natural letters and numbers as well as custom graphic symbols. Figure 16 shows a typical 5x7 dot matrix character set.

The third type of display is most commonly used in conjunction with the previous types. A function indicator or icon provides status information about the system. They are only capable of being turned on or off. One example would be a digital multimeter. The meter has three 1/2 digits which are 7-segment type and also some icons for volts, amps, ohms and the ranges for m, µ, K, and M. Another example would be a cellular telephone. The LCD panel will have eight or more 5x7 dot matrix characters with icons for events such as in use, roam, no service, battery status, and signal strength. Figure 17 shows what a typical cellular phone panel might resemble.

3

FIGURE 16: 5x7 DOT MATRIX DISPLAY

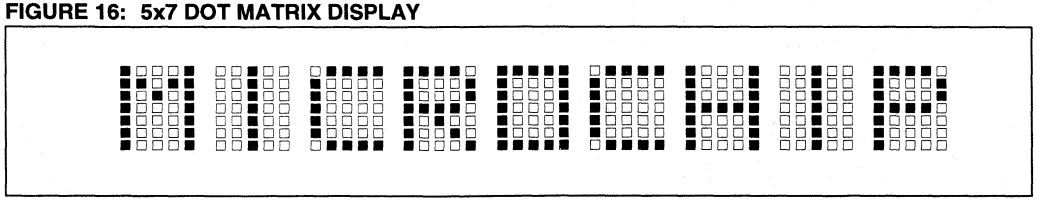
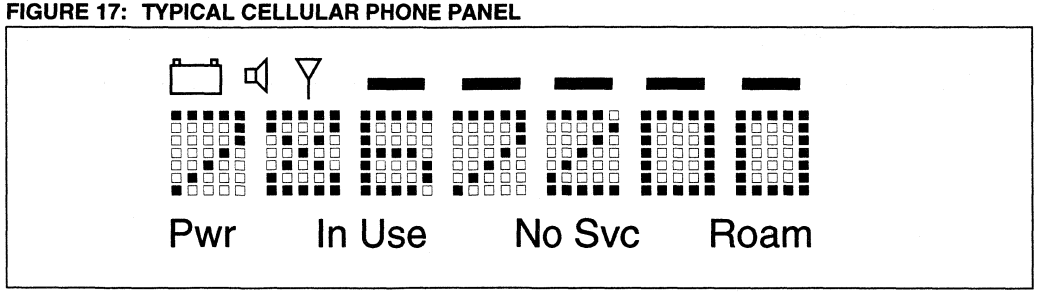


FIGURE 17: TYPICAL CELLULAR PHONE PANEL



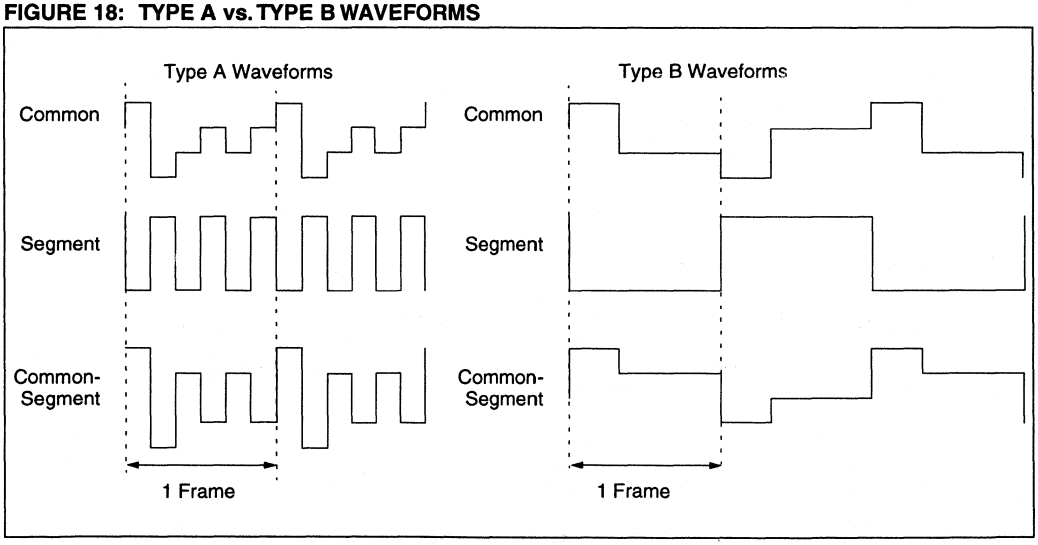
© 1997 Microchip Technology Inc.

HOW ARE LCD PANELS DRIVEN?

So far, the mysteries of how an LCD is made, how it works, and what the different type of panels have been revealed. This section will demystify the LCD waveforms. An LCD can be characterized by the MUX ratio and bias, but one piece of information is still missing - Drive Waveforms. LCDs can be driven by two types of waveforms: Type A and Type B. Before the definitions of the two types are given, the term frame frequency must be defined. The LCD frame frequency is the rate at which the backplane and segment outputs change. The frame frequency is then calculated to be the LCD period / 2 • number of backplanes. The range of frame frequencies is from 25 to 250 Hz with the most common being between 50 and 150 Hz. Higher frequencies result in higher power consumption while lower frequencies cause flicker in the images on the LCD panel. An earlier section mentioned that a LCD driver must maintain a 0 VDC potential across each pixel. Type A waveforms maintain 0 VDC over a single frame whereas Type B takes two frames. Figure 18 shows both types of waveforms with 1/3 MUX and 1/3 Bias. PIC16C92X microcontrollers support only Type A waveforms.

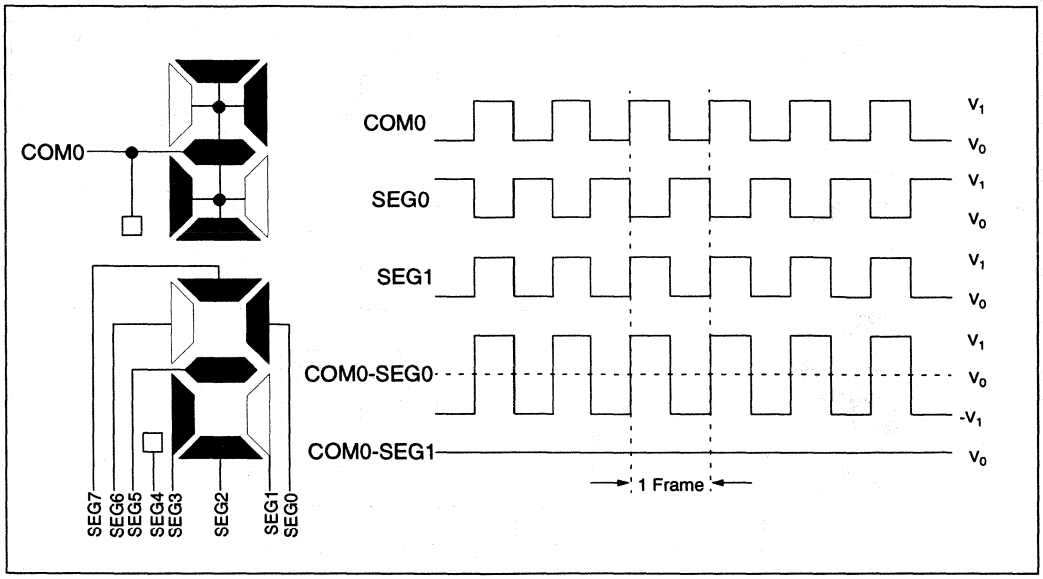
The voltage applied across a particular pixel is the voltage on the COM pin minus the voltage on the SEG pin. If the resulting voltage is at or above the VON threshold then the pixel is visible. Otherwise the voltage will be at or below the VOFF threshold and the pixel will not be visible. This formula is used for all drive/bias methods. The following figures show each of the modes that are currently supported by the PIC16C92X devices. Since the PIC16C92X devices only support Type A waveforms, only Type A waveforms for each of the modes are shown. Each figure has the LCD period and the frame locations marked.

FIGURE 18: TYPE A vs. TYPE B WAVEFORMS



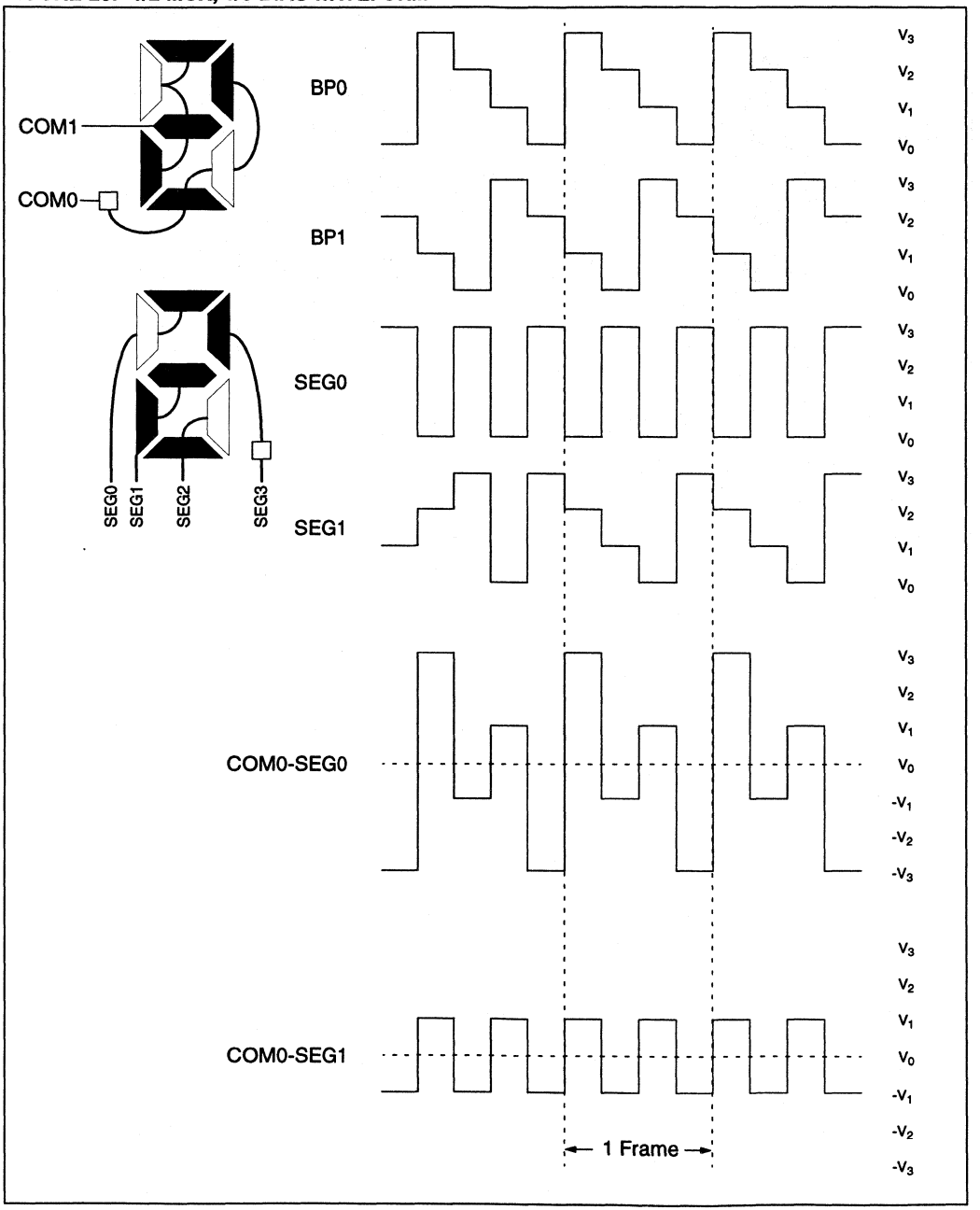
© 1997 Microchip Technology Inc.

FIGURE 19: STATIC WAVEFORMS



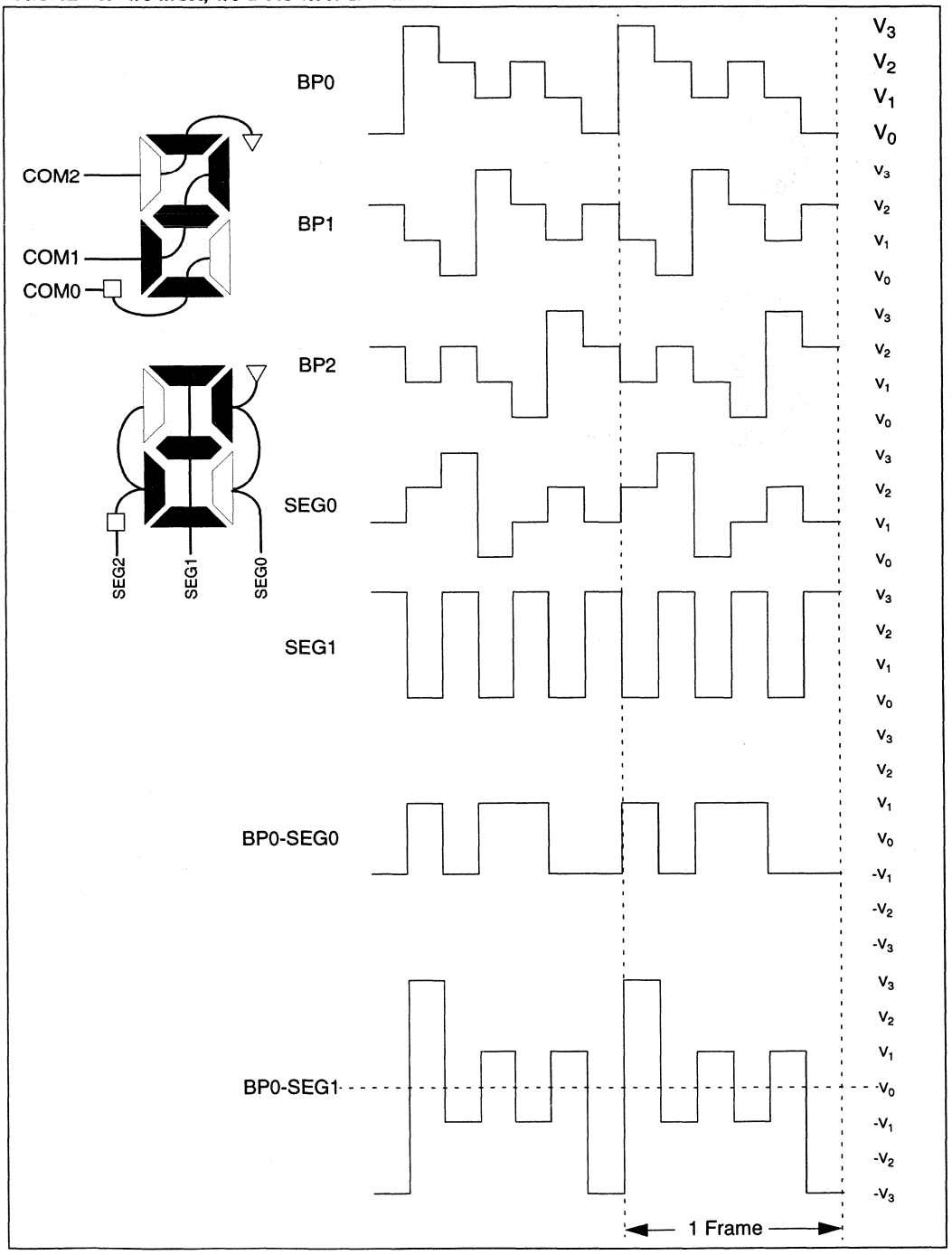
© 1997 Microchip Technology Inc.

FIGURE 20: 1/2 MUX, 1/3 BIAS WAVEFORM



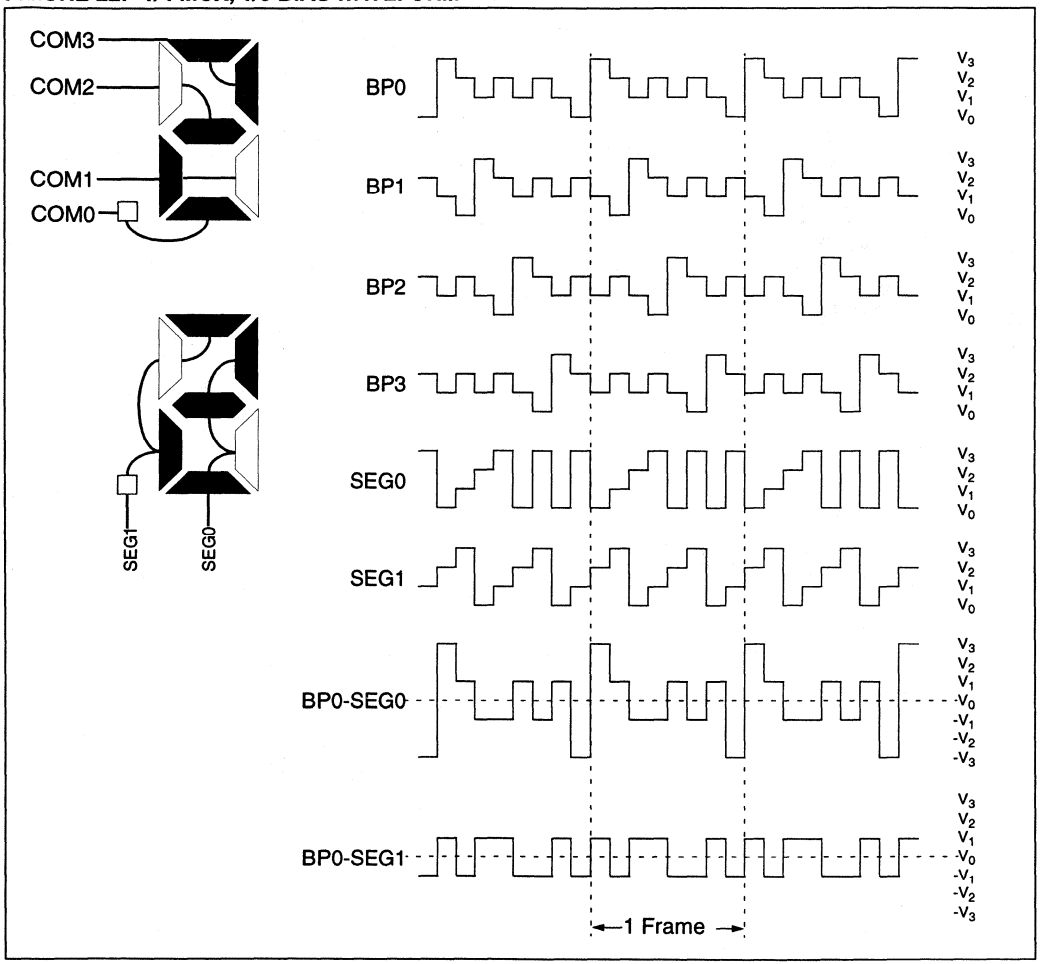
© 1997 Microchip Technology Inc.

FIGURE 21: 1/3 MUX, 1/3 BIAS WAVEFORM



© 1997 Microchip Technology Inc.

FIGURE 22: 1/4 MUX, 1/3 BIAS WAVEFORM



© 1997 Microchip Technology Inc.

DISCRIMINATION RATIO

Now that the LCD waveforms have been presented, let's calculate the discrimination ratio for some of them. The first example is a static waveform from Figure 19. The voltages V_1 and V_0 will be assigned values of 1 and 0. The next step is to construct a matrix for one frame to help visualize the DC and RMS voltages present on an individual pixel when it is ON and when it is OFF. The rest of the following shows the calculation of the DC, RMS, and Discrimination Ratio.

EXAMPLE 1: DISCRIMINATION RATIO CALCULATION FOR STATIC MUX

BPx	0	1	
SEGx	1	0	ON
	0	1	OFF

BPx - SEGx [ON] = -1 + 1, $V_{DC} = 0$

BPx - SEGx [OFF] = 0 + 0, $V_{DC} = 0$

$$V_{RMS}\,[ON] = \Delta V\sqrt{\frac{(-1)^2 + (1)^2}{2}} = 1\Delta V$$

$$V_{RMS}\,[OFF] = \Delta V\sqrt{\frac{(0)^2 + (0)^2}{2}} = 0\Delta V$$

$$D = \frac{V_{RMS}\,[ON]}{V_{RMS}\,[OFF]} = \frac{1\Delta V}{0\Delta V} = \infty$$

The next example is for Figure 22 which is a 1/4 MUX, 1/3 BIAS waveform. For this example, the values 3, 2, 1 and 0 will be assigned to V_3, V_2, V_1, and V_0 respectively. The frame matrix, DC voltage, RMS voltage and discrimination ratio calculations are shown in Example 2:

EXAMPLE 2: DISCRIMINATION RATIO CALCULATION 1/4 MUX

BP0	0	3	2	1	2	1	2	1	
BP1	2	1	0	3	2	1	2	1	
BP2	2	1	2	1	0	3	2	1	
BP3	2	1	2	1	2	1	0	3	
SEGx	3	0	3	0	3	0	3	0	ON
	1	2	1	2	1	2	1	2	OFF

BP0 - SEGx [ON] = - 3 + 3 - 1 + 1 - 1 + 1 - 1 + 1 $V_{DC} = 0$

BP0 - SEGx [OFF] = - 1 + 1 - 1 + 1 - 1 + 1 - 1 + 1 $V_{DC} = 0$

$$V_{RMS}\,[ON] = \Delta V\sqrt{\frac{(-3)^2 + (3)^2 + (-1)^2 + (1)^2 + (-1)^2 + (1)^2 + (-1)^2 + (1)^2}{8}} = \sqrt{3}\,\Delta V$$

$$V_{RMS}\,[OFF] = \Delta V\sqrt{\frac{(-1)^2 + (1)^2 + (-1)^2 + (1)^2 + (-1)^2 + (1)^2 + (-1)^2 + (1)^2}{8}} = \Delta V$$

$$D = \frac{V_{RMS}\,[ON]}{V_{RMS}\,[OFF]} = \frac{\sqrt{3}\,\Delta V}{1\ \Delta V} = 1.732$$

As shown in these examples, static displays have excellent contrast. The higher the multiplex ratio of the LCD, the lower the discrimination ratio, and therefore, the lower the contrast of the display.

The following table shows the VOFF, VON and discrimination ratios of the various combinations of MUX and BIAS.

TABLE 5: DISCRIMINATION RATION vs. MUX AND BIAS

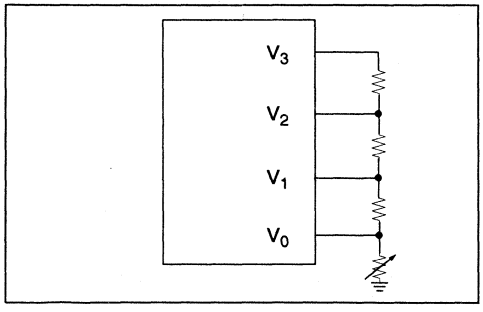
	1/3 BIAS		
	Voff	**Von**	**D**
STATIC	0	1	∞
1/2 MUX	0.333	0.745	2.236
1/3 MUX	0.333	0.638	1.915
1/4 MUX	0.333	0.577	1.732

Table 5 shows that as the multiplex of the LCD panel increases, the discrimination ratio decreases. The contrast of the panel will also decrease, so to provide better contrast the LCD voltages must be increased to provide greater separation between each level.

LCD VOLTAGE GENERATION

Among the many ways to generate LCD voltage, two methods stand out above the crowd: resistor ladder and charge pump.

FIGURE 23: RESISTOR LADDER

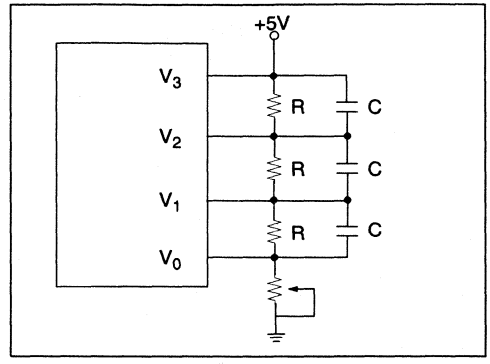


The resistor ladder methods, shown in Figure 23 is most commonly used for higher Vcc voltages. This method uses inexpensive resistors to create the multilevel LCD voltages. Regardless of the number of pixels that are energized the current remains constant. The voltage at point V3 is typically tied to Vcc, either internally or externally.

The resistance values are determined by two factors: display quality and power consumption. Display quality is a function of the LCD drive waveforms. Since the LCD panel is a capacitive load, the waveform is distorted due to the charging and discharging currents. This distortion can be reduced by decreasing the value of resistance. However this change increases the power consumption due to the increased current now flowing through the resistors. As the LCD panel increases in size, the resistance value must be decreased to maintain the image quality of the display.

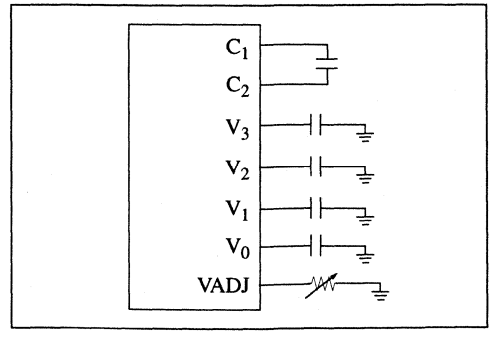
Sometimes the addition of parallel capacitors to the resistance can reduce the distortion caused by charging/discharging currents. This effect is limited since at some point a large resistor and large capacitor cause a voltage level shift which negatively impacts the display quality. In general, R is 1 kΩ to 50 kΩ and the potentiometer is 5 kΩ to 200 kΩ.

FIGURE 24: RESISTOR LADDER WITH CAPACITORS



A charge pump is ideal for low voltage battery operation because the VDD voltage can be boosted up to drive the LCD panel. The charge pump requires a charging capacitor and filter capacitor for each of the LCD voltages as seen in Figure 25. These capacitors are typically polyester, polypropylene, or polystyrene material. Another feature that makes the charge pump ideal for battery applications is that the current consumption is proportional to the number of pixels that are energized.

FIGURE 25: CHARGE PUMP



© 1997 Microchip Technology Inc.

CONTRAST

Although contrast is heavily dependent on the light source available and the multiplex mode, it also varies with the LCD voltage levels. As previously seen, a potentiometer is used to control the contrast of the LCD panel. The potentiometer sets the separation between each of the LCD voltages. The larger the separation, the better the contrast achievable.

CONCLUSION:

Hopefully you can now answer the questions:

- What are the basic components in an LCD panel?
- How does an LCD work?
- What are the different types of LCD panels?
- How are LCD panels driven?

This application note has covered LCD fundamentals in great detail. Please refer to the PIC16C92X microcontroller data sheet for more information. Also application note AN649, "Yet Another Clock Featuring the PIC16C924" shows an example application using the PIC16C924.

© 1997 Microchip Technology Inc.

APPENDIX A: LIST OF LCD MANUFACTURERS AND DISTRIBUTORS

AEG-MIS
3340 Peachtree Rd. NE Suite 500
Atlanta, GA 30326
TEL: 404-239-0277
FAX: 404-239-0383

All Shore INDS Inc.
1 Edgewater Plaza
Staten Island, NY 10305
TEL: 718-720-0018
FAX: 718-720-0225

Crystaloid
5282 Hudson Drive
Hudson, OH 44236-3769
TEL: 216-655-2429
FAX: 216-655-2176

DCI Inc.
14812 W. 117th St.
Olathe, KS 66062-9304
TEL: 913-782-5672
FAX: 913-782-5766

Excel Technology International Corporation
Unit 5, Bldg. 4, Stryker Lane
Belle Mead, NJ 08502
TEL: 908-874-4747
FAX: 908-874-3278

F-P Electronics/Mark IV Industries
6030 Ambler Drive
Mississauga, ON Canada L4W 2PI
TEL: 905-624-3020
FAX: 905-238-3141

Hunter Components
24800 Chagrin Blvd, Suite 101
Cleveland, OH 44122
TEL: 216-831-1464
FAX: 216-831-1463

Satori Electric
23717 Hawthorne Blvd. 3rd Floor
Torrance, CA 90505
TEL: 310-214-1791
FAX: 310-214-1721

Interstate Electronics Corp.
1001 E. Bull Rd.
Anaheim, CA 92805
TEL: 800-854-6979
FAX: 714-758-4111

Kent Display Systems
343 Portage Blvd.
Kent, OH 44240
TEL: 330-673-8784

LCD Planar Optics Corporation
2100-2 Artic Ave.
Bohemia, NY 11716
TEL: 516-567-4100
FAX: 516-567-8516

LXD Inc.
7650 First Place
Oakwood Village, OH 44146
TEL: 216-786-8700
FAX: 216-786-8711

Nippon Sheet Glass
Tomen America Inc.
1285 Avenue of the Americas
New York, NY 10019
TEL: 212-397-4600
FAX: 212-397-3351

OPTREX America
44160 Plymouth Oaks Blvd.
Plymouth, MI 48170
TEL: 313-416-8500
FAX: 313-416-8520

Phillips Components
LCD Business Unit
1273 Lyons Road, Bldg G
Dayton, OH 45459
TEL: 573-436-9500
FAX: 573-436-2230

Varitronix Limited Inc.
3250 Wilshire Blvd. Suite 1901
Los Angeles, CA 90010
TEL: 213-738-8700
FAX: 213-738-5340

© 1997 Microchip Technology Inc.

Seiko Instruments USA Inc.
Electronic Components Division
2990 West Lomita Blvd.
Torrance, CA 90505
TEL: 213-517-7770
213-517-8113
FAX: 213-517-7792

Standish International
European Technical Center
Am Baümstuck II
65520 Bad Camberg/Erbach
Germany
TEL: 011 49 6434 3324
FAX: 011 49 6434 377238

Standish LCD
W7514 Highway V
Lake Mills, WI 53551
TEL: 414-648-1000
FAX: 414-648-1001

Truly Semiconductors Ltd. (USA)
2620 Concord Ave.
Suite 106
Alhambra, CA 91803
TEL: 818-284-3033
FAX: 818-284-6026

Truly Semiconductor Ltd.
2/F, Chung Shun Knitting Center
1-3 Wing Yip Street,
Kwai Chung, N.T., Hong Kong
TEL: 852 2487 9803
FAX: 852 2480 0126

Varitronix Limited Inc.
4/F, Liven House
61-63 King Yip Street
Kwun Tong, Kowloon
Hong Kong
TEL: 852 2389 4317
FAX: 852 2343 9555

Varitronix (France) S.A.R.L.
13/15 Chemin De Chilly
91160 Champlan
France
TEL: (33) 1 69 09 7070
FAX: (33) 1 69 09 0535

Varitronix Italia, S.R.L.
Via Bruno Buozzi 90
20099 Sesto San Giovanni
Milano, Italy
TEL: (39) 2 2622 2744
FAX: (39) 2 2622 2745

Varitronix (UK) Limited
Display House, 3 Milbanke Court
Milbanke Way, Bracknell
Berkshire RG12 1BR
United Kingdom
TEL: (44) 1344 30377
FAX (44) 1344 300099

Varitronix (Canada) Limited
18 Crown Steel Drive, Suite 101
Markham, Ontario
Canada L3R 9X8
TEL: (905) 415-0023
FAX: (905) 415-0094

Vikay America Inc.
195 W. Main St.
Avon, CT 06001-3685
TEL: 860-678-7600
FAX: 860-678-7625

© 1997 Microchip Technology Inc.

DISTRIBUTORS

Allied Electronics Inc.
7410 Pebble Drive
Fort Worth, TX 76118
TEL: 800-433-5700
http://www.allied.avnet.com

Digikey Corporation
701 Brooks Ave. South
Thief River Falls, MN 56701-0677
TEL: 800-344-4539
http://www.digikey.com

Newark Electronics
Administrative Office
4801 N. Ravenswood Ave.
Chicago, IL 60640-4496
TEL: 312-784-5700
FAX: 312-907-5217

© 1997 Microchip Technology Inc.

Smart Battery Charger with SMBus Interface

Authors: *Laszlo Kiraly, Linear Technology*
Robert Schreiber, Microchip Technology

INTRODUCTION

This application note provides the schematics, software listings, and circuit board layout for a PIC16C73 based Smart Battery Charger. The Demo Board, DC101, is available to selected customers through Linear Technology Corp. product marketing.

The DC101 (Figure 1) is the Smart Battery Charger (SBC) portion of a Smart Battery System. A simplified block diagram of a Smart Battery System is shown in Figure 3 (refer to "The Smart Battery Charger Specification," Duracell).

The Smart Battery (SB) contains circuitry which provides charging information to the SBC. The SBC receives this information in terms of ChargingVoltage(), ChargingCurrent(), and AlarmWarning() commands from the SB. The SBC and SB communicate via the System Management Bus (SMB), which is an implementation of the I^2C bus (refer to "The Smart Management Bus Specification," Intel; and "The I^2C Bus and How to Use It," Philips Semiconductor). The SBC sets the charging current and voltage based on input from the SB. The SBC charger also has the intelligence to monitor the SB's thermistor. The thermistor provides temperature information for charge termination, and battery chemistry information.

3

FIGURE 1: DC101 SMART BATTERY CHARGER

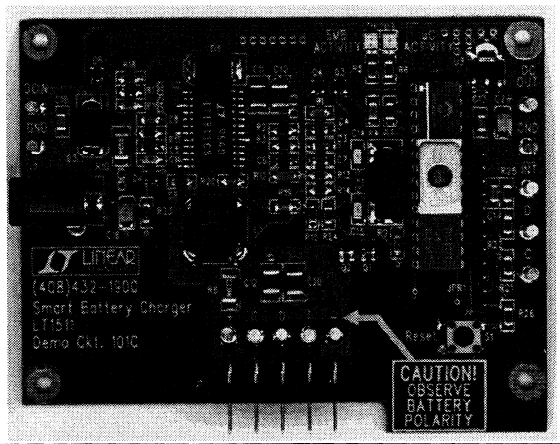
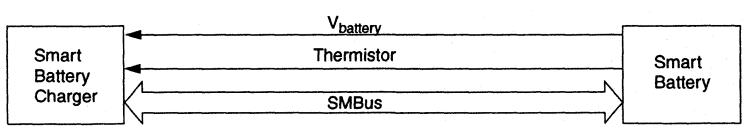
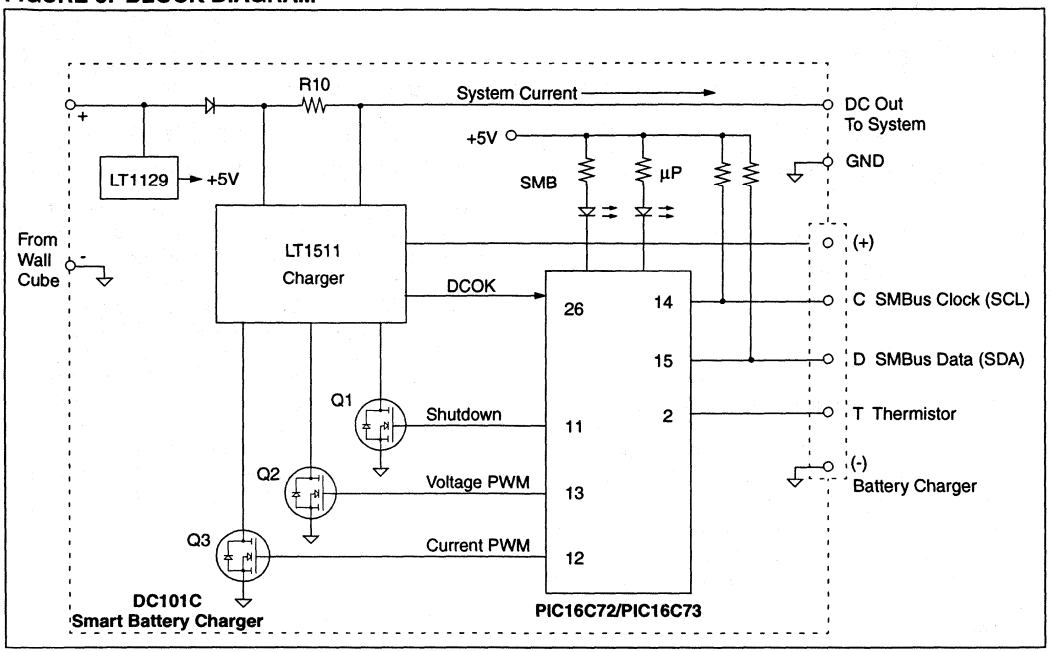


FIGURE 2: SIMPLIFIED BLOCK DIAGRAM



© 1997 Microchip Technology Inc.

FIGURE 3: BLOCK DIAGRAM



OPERATING THE BOARD

Input voltage. The nominal input voltage of the board is 24V DC (Refer to Appendix A for Performance Summary). The input voltage must be higher than the battery voltage by a minimum of 3V. The minimum input voltage is 16V, limited by the undervoltage lockout circuit in the LT1511 and set by resistors R16, R17 and R18. The highest input voltage is 27.4V, that is limited by the maximum input voltage of the LT1511. The input is protected against reverse polarity up to 30V.

Input current. The sum of the system current and the charger input current is limited by the LT1511 to 2A. When both the system current and the charger input current requirements are high, the charger reduces the charging current to meet the 2A current limit.

RED LED. It indicates SMBus activity. It lights-up for about 1 second when the charger recognizes its own address (12hex) on the SMBus.

Green LED. Flashing green LED indicates microprocessor activity and charger status.

Fast blinking (approximately 8 Hz) indicates normal microprocessor activity and either trickle charge or shut down charger status. After valid voltage and current data have been received the blinking speed of the LED slows down to about 2 Hz, indicating normal charging.

Battery removal, thermistor measurements. The charger periodically checks the thermistor in the battery through the T terminal. When the thermistor is out of normal operating temperature range, the charger switches to trickle charge mode and increases the flashing frequency of the green LED to indicate an abnormal charging condition. When the resistance of the thermistor is in the 500Ω < Rth < 1.5 kΩ range, the charger assumes Li Ion battery is at the output, and instead of trickle charging the battery it shuts-down the charger until a valid voltage and current request arrives from the battery.

An open thermistor forces the charger into trickle charge mode and the charger disregards data on the SMBus.

The schematics, parts list, and circuit layout are shown in Appendix A.

© 1997 Microchip Technology Inc.

SYSTEM MANAGEMENT BUS (SMBUS)

When charge in the Smart Battery (SB) drops below 85% of the nominal capacity, it initiates communication over the SMBus every 64 seconds. After sending a START sequence the battery addresses the Smart Battery charger and waits for acknowledgment (ACK) from it. If the charger fails to acknowledge the word, the battery terminates further communication by placing a STOP sequence onto the SMBus. If the charger acknowledges (ACK) the reception of first word, the battery continues the communication sequence and it sends six more words to the charger. The complete current and voltage request communication sequence is shown below:

```
START
address (12 hex)
ChargingCurrent() command code (14 hex),
current_LSB
current_MSB,
address (12 hex)
ChargingVoltage() command code (15 hex)
voltage_LSB
voltage_MSB
STOP
```

The idealized SMBus waveforms shown in Figure 4 illustrate SMBus communication between the battery and the charger. The first seven bits after the start sequence is the battery address. The R/$\overline{W}$ tells the charger that the battery attempts to write to the charger.

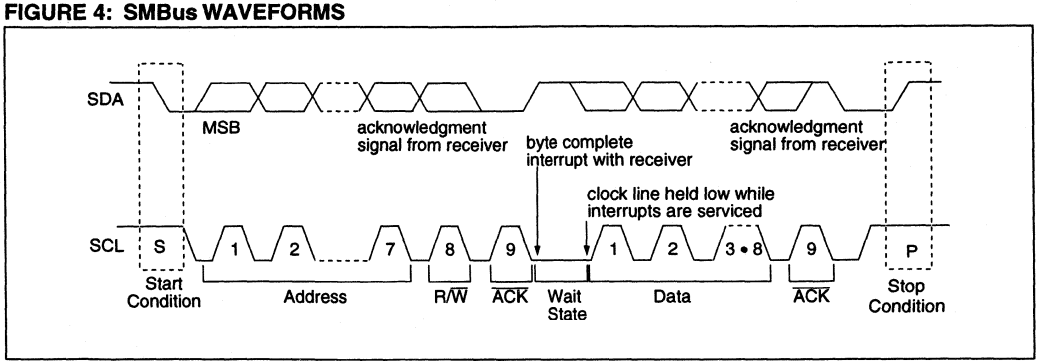
During the acknowledge period (ACK), the charger becomes active and it pulls the data line (SDA) low indicating reception of a data word. When reception of the data word is not acknowledged by the charger, the battery terminates the communication by sending a STOP sequence to the bus. The clock pulses for the communication are always generated by the battery (BUS MASTER).

CONCLUSION

The PIC16C73 contains the on-board peripherals necessary to easily implement an SMBus battery charger. The I^2C module allows for the SMBus communications and the PWM modules allow for voltage and current control. This high level of integration reduces the external components required and increases the flexibility of the design.

A complete software listing is shown in Appendix B.

FIGURE 4: SMBus WAVEFORMS



© 1997 Microchip Technology Inc.

APPENDIX A:

TABLE A-1: PERFORMANCE SUMMARY

Parameter	Conditions	MIN	TYP	MAX	UNIT
Input Voltage		16.0	24.0	27.0	V
Input Current	Hardware Limited			2.0	A
Output Current	Software Limited	0		2.5	A
Output Voltage	Software Limited	4		20	V
Thermistor Resistance	NiMH Battery	3k		30k	Ohms
	Li Ion	500	1000	1500	Ohms

© 1997 Microchip Technology Inc.

FIGURE A-1: SCHEMATIC DIAGRAM

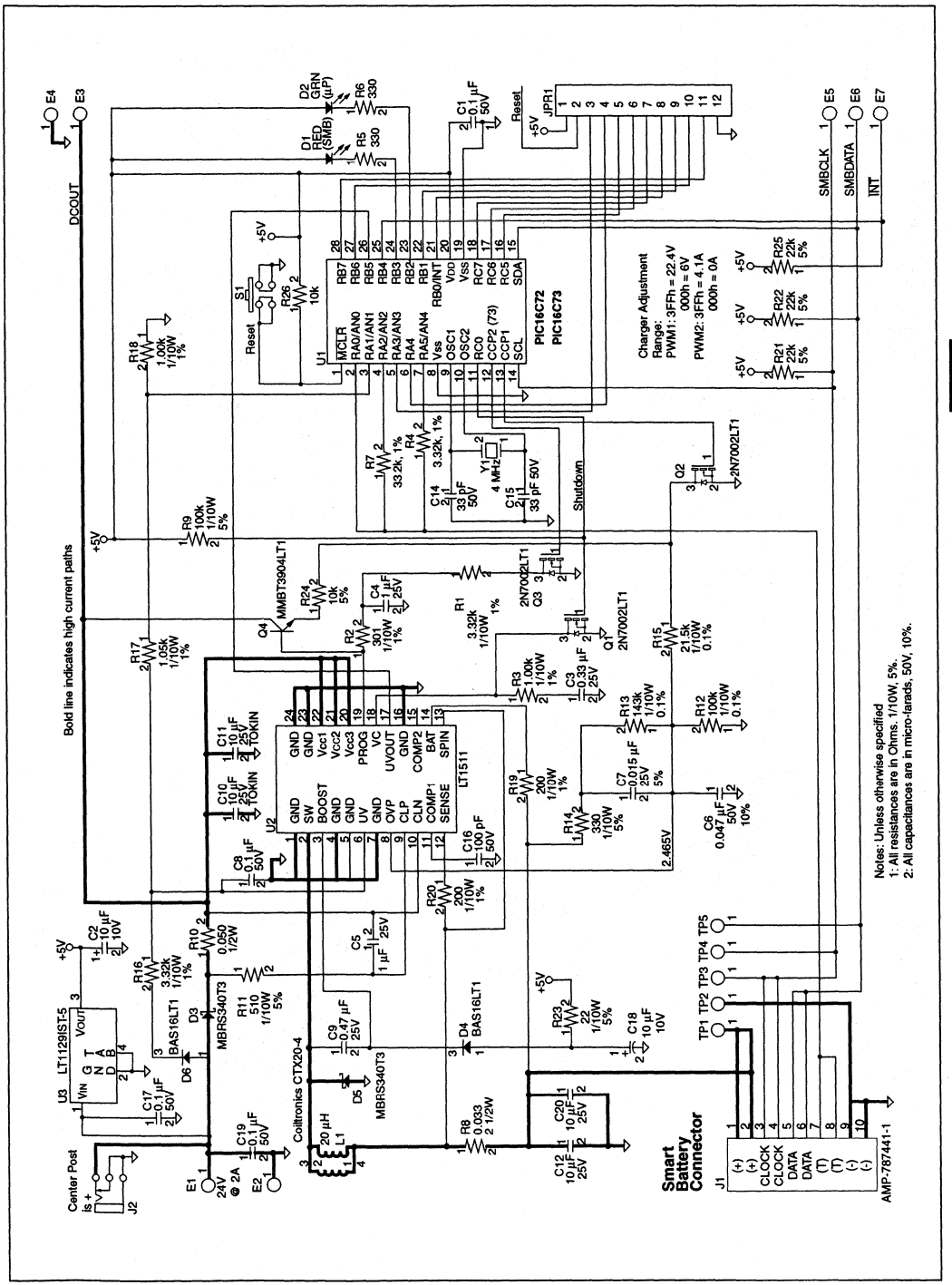


TABLE A-2: PARTS LIST

Item	Quantity	Reference	Part Description	
1	4	C1,C8,C17,C19	CAP., CHIP 0.1 µF 50V	AVX 12065C104KATMA
2	2	C2,C18	CAP., TANT. 10 µF 10V	AVX TAJB106M010
3	1	C3	CAP., CHIP 0.33 µF 25V	AVX 12063G334ZAT2
4	2	C4,C5	CAP., CHIP 1 µF 25V	AVX 12063G105ZATMA
5	1	C6	CAP., CHIP 0.047 µF 50V 10%	AVX 12065C473KAT
6	1	C7	CAP., CHIP 0.015 µF 25V 5%	AVX 12063C153KAT2
7	1	C9	CAP., CHIP 0.47 µF 25V	AVX 12063G474ZAT2
8	4	C10,C11,C12,C20	CAP., CERAMIC 10 µF 25V	MARCON THC550EIE106Z
10	2	C14,C15	CAP., CHIP 33 pF 50V	AVX 1206A330KAT2
11	1	C16	CAP., CHIP 100 pF 50V	VITRAMON VJ1206A101KXA
12	1	D1	LED, SF1-BR RED	DATA DISPLAY PRODUCT
13	1	D2	LED, SF1-G GRN	DATA DISPLAY PRODUCT
14	2	D3,D5	DIODE	MOTOROLA MBRS340T3
15	2	D4,D6	DIODE	MOTOROLA BAS16LT1
16	12	TP1--TP5, E1--E7	TESTPOINT, TURRET	KEYSTONE 1502-2
17	1	JPR1	JUMPER, HEADER 12 PIN	COMM CON CONN. 3801S-12-G1
18	1	J1	CONNECTOR,	AMP-787441-1
19	1	J2	CONNECTOR,	CUI-PJ-002A CUI-STACK
20	1	L1	INDUCTOR, 20 µH	COILTRONICS CTX20-4
21	3	Q1,Q2,Q3	MOSFET N, Channel	MOTOROLA 2N7002LT
22	1	Q4	TRANS. NPN,	MOTOROLA MMBT3904LT1
23	1	R2	RES., 301, 1/8W, 1%	AVX CR32-3010F-T
24	2	R3,R18	RES., 1.00k, 1/8W, 1%	AVX CR32-1001F-T
25	3	R1,R4,R16	RES., 3.32k, 1/8W, 1%	AVX CR32-3321F-T
26	3	R5,R6,R14	RES., 330, 1/8W, 5%	AVX CR32-331J-T
27	1	R7	RES., 33.2k, 1/8W, 1%	BECKMAN BCR1/8-3322F-T
28	1	R8	RES., 0.033, 1/2W, 5%	IRC LR2010-01-R033-J
29	1	R9	RES., 100k, 1/8W, 5%	AVX CR32-104J-T
30	1	R10	RES., 0.050, 1/2W, 5%	IRC LR2010-01-R050-J
31	1	R11	RES., 510, 1/8W, 5%	AVX CR32-511J-T
32	1	R12	RES., 100k, 1/8W, 0.1%	IRC W1206R-03-1003-B
33	1	R13	RES., 143k, 1/10W, 0.1%	IRC W1206R-03-1433-B
34	1	R15	RES., 21.5k, 1/10W, 0.1%	IRC W1206R-03-2152-B
35	1	R17	RES., 1.05k, 1/8W, 1%	AVX CR32-1051F-T
36	2	R20,R19	RES., 200, 1/8W, 1%	AVX CR32-2000F-T
37	3	R21,R22,R25	RES., 22k, 1/10W, 5%	AVX CR-32-223J-T
38	1	R23	RES., 22, 1/10W, 5%	AVX CR32-220J-T
39	2	R24,R26	RES., 10k, 1/8W, 5%	DALE CR1206-103J
40	1	S1	PB-SWITCH, MJTP1230	MORS-ASC MJTP1230
41	1	U1	I.C., PIC16C73, PIC16C72	MICROCHIP IC, Microcontroller
42	1	U2	I.C., LT1511	LINEAR TECHNOLOGY IC, Battery Charger
43	1	U3	I.C., LT1129IST-5	LINEAR TECHNOLOGY IC, Voltage Regulator
44	1	Y1	CRYSTAL, 4 MHz	EPSON(USA) MA-505-4.00M-C2
45	1	XU1	I.C., SOCKETS	COMM CON 7167-14-G2

© 1997 Microchip Technology Inc.

FIGURE A-2: CIRCUIT COMPONENT SIDE

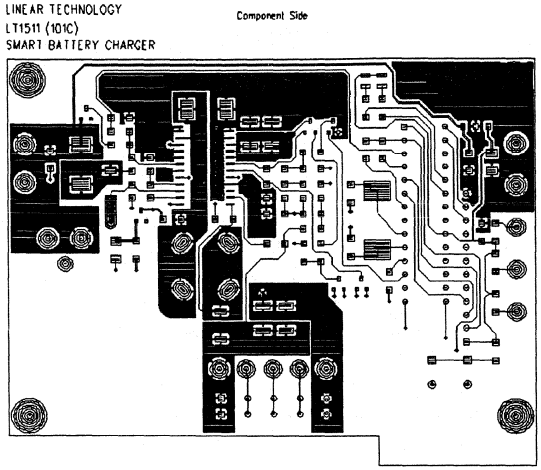
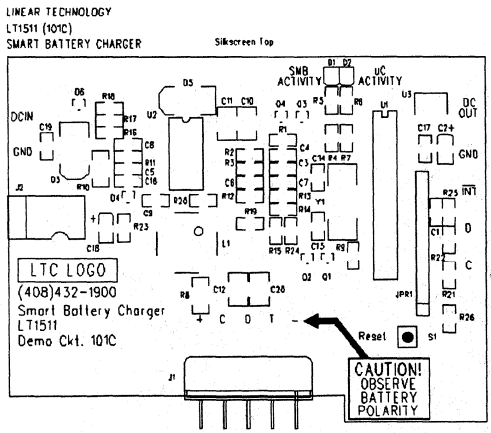


FIGURE A-3: COMPONENT SIDE SILKSCREEN



3

FIGURE A-4: COMPONENT SIDE SOLDERSIDE

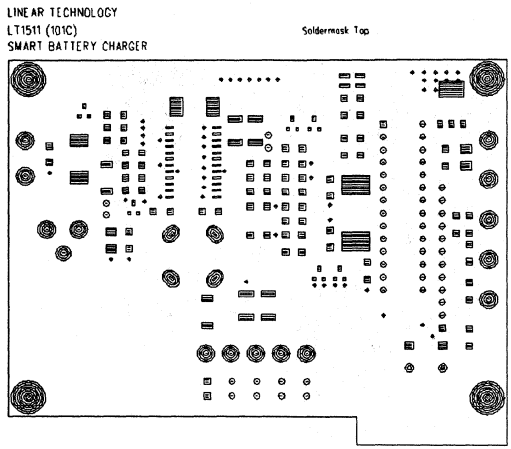
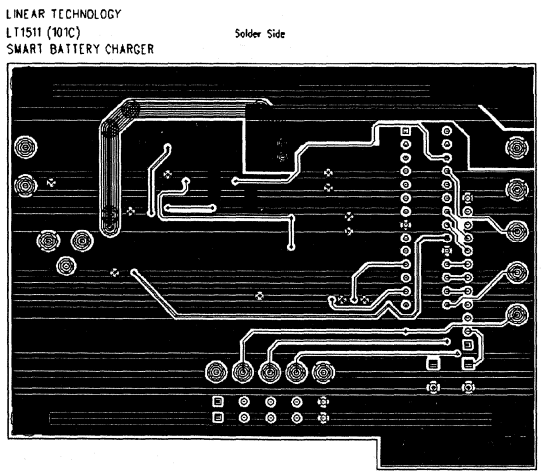
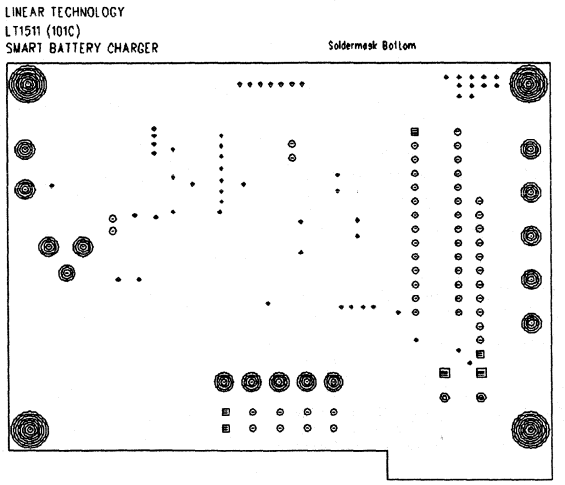


FIGURE A-5: CIRCUIT SOLDER SIDE



© 1997 Microchip Technology Inc.

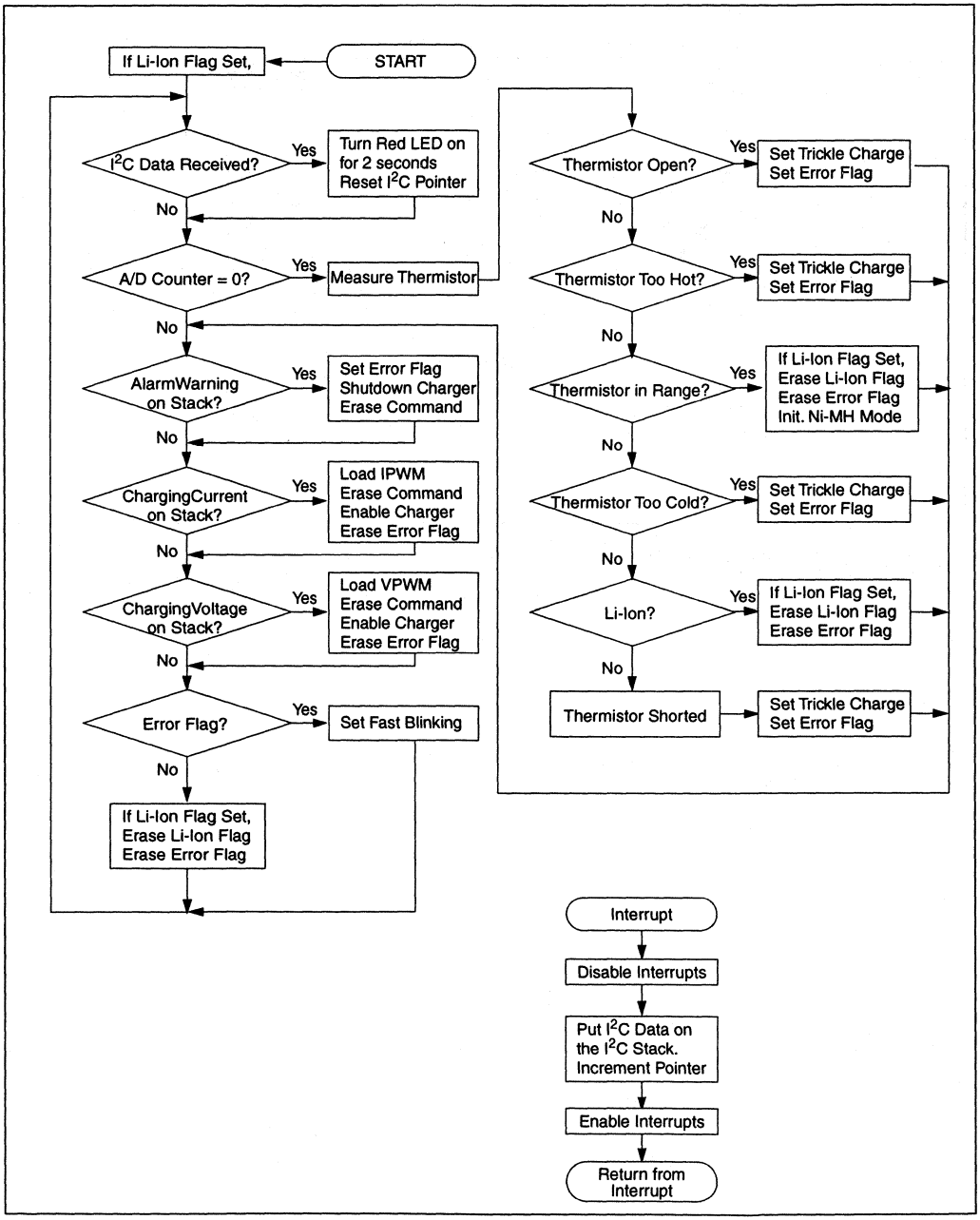
FIGURE A-6: SOLDER SIDE SOLDER MASK



© 1997 Microchip Technology Inc.

APPENDIX B:

FIGURE B-1: SOFTWARE FLOWCHART



© 1997 Microchip Technology Inc.

```
//-------------------------------------------------------------------------
//                        CHRGR101.C                        Version: 1.01
//                     By: Laszlo Kiraly
//                   Email:KiralyL@aol.com
//                    Linear Technology
//                  Applications Department
//          1630 McCarthy Blvd, Milpitas, CA 95035, USA
//          Phone: (408) 432-1900,  Fax: (408) 434-0507
/*-------------------------------------------------------------------------
                                                    April 16,1996
This program:
 Sets-up PWMs and I2C communication.
 It receives data from I2C (including its own address) and stores them
 in i2c_data[10] as they come in, including its own address (0x12).

               0 charger i2c address (always 0x12)
               1 ChargingCurrent() CMD (0x14)
               2 charging current data byte L
               3 charging current data byte H       (unsigned int, mA)
               4 charger i2c address (0x12)
               5 ChargingVoltage CMD (0x15)
               6 charging voltage data byte L
               7 charging voltage data byte H       (unsigned int, mV)

Revision changes:

- scales current word and loads value to IPWM (PWM2).
- scales voltage word and loads value to VPWM (PWM1).
- 16CC73.H bits for CCP1CON registers need to be specified.
- thermistor limits were changed   (2/21/96)
- using easy math (2/21/96)
//
- comments were added                                          (4/11/96)
- init() function was created                                  (4/11/96)
- AlarmWarning() function now checks b15:b12 bits.             (4/11/96)
* At LT1511 UV shutdown (DCOK-L) uP pulls its shut-down high.
- The uP enables the charger after valid data received from battery.(4/11/96)
- No broadcast from battery timeout (180 sec. nom.) now implemented.(4/11/96)
- SMBus reset function has been added.                         (4/15/96)
- GIE disable and enable was removed (__int handles them)      (4/16/96)
-------------------------------------------------------------------------
*/
#include "16C73a.H"
#include "math.h"
//
void delay( void );                     // 840msec delay for red LED
void delay1( void);                     // 20ms delay for while() loop
void delay_5us( void );
void timer0( char );
void load_ipwm( void );                 // scales received value and loads PWM1
void load_vpwm( void );                 // scales ChargingCurrent, loads PWM2
void initiv( char );                    // sets voltage and current PWMs
void ad_th( void );                     // measures thermistor, controls charger
void init_var( void );                  // initializes variables (general)
void clear_smbus( void );               // sends start-stop sequence to SMBus
//
char i2c_data[10];                      // i2C stack
char i2c_counter;                       // i2c stack pointer
//
bits  flag;
                                        // flag.0 is set on return from I2C INT.
                                        // flag.1 blinking speed 1 = high speed
                                        // flag.2 themistor/resistor out of range
                                        // flag.3=1 comm. timeout, inibits cntr
char LED_counter, ad_counter;
long unsigned ad_val;
```

```
long unsigned com_timeout_cntr;
long unsigned clear_smbus_cntr;
;
char val;
//
//---------------------------------------------------------------------------
void __INT( void)
{
  SSPCON.CKP = 0;                       // HOLD CK 0
  flag.0 = 1;

  i2c_data[i2c_counter] = SSPBUF;
  i2c_counter++;
  if(i2c_counter > 7)
  {
     i2c_counter=0;                     // reset counter if overflows
  }
  PIR1.SSPIF = 0;
  SSPCON.CKP = 1;                       // release ck
}
//
// ----------------------------------------------------------
void main()
{
  init_var();                           // initializes variables (hardware setup)
  //
  // timer0(150);
  //
  delay();                              // wait 1sec until voltages settle
  //
  i2c_counter =0;                       // clear i2c data pointer
  ad_counter = 5;                       // clear
  //
  com_timeout_cntr = 0;
  clear_smbus_cntr = 0;                 // sets time between SMBus inits.
  //
  SSPADD = 0x12;                        // define slave address
  //
  flag = 0;
  flag.1 = 1;                           // set fast blinking
  flag.2 = 1;                           // thermistor out of both NiMH and liIon
  flag.3 = 1;                           // disable com_timeout_cntr
  //
  PIE1.SSPIE  = 1;                      // enable i2c interrupt
  INTCON.PEIE = 1;                      // enable peripheral INTS
  INTCON.GIE  = 1;                      // general INT enable
  //
  while(1)                              // infinite loop in MAIN
  {
    delay1();
    //
    // ------------------------------------------------------------
    if(com_timeout_cntr == 6250)        // no communication timeout 100-> 2.88sec
    {
       flag.1 = 1;                      // set fast blinking
       flag.3 = 1;                      // communication timeout cntr disabled
       //
       PORTC.0 = 1;                     // disable charger
       com_timeout_cntr = 0;            // reset timer
       //
       PORTB.3 = 0;                     // turn red LED on for 50ms
       delay1();
       PORTB.3 = 1;                     // red LED off
    }
    //
    if( !flag.3 ) com_timeout_cntr++;   // if no timeout run counter
```


© 1997 Microchip Technology Inc.

```
else com_timeout_cntr = 0;
if( flag.2) flag.3 = 1;              // if th. out of range reset cntr
                                     // do not override th. based decisions

//
// ---------------------------------------------------------------------
if( flag.3) clear_smbus_cntr++;      // clear SMBus
if( clear_smbus_cntr == 1000)
{
  clear_smbus();
  clear_smbus_cntr = 0;
}
if( flag.2) clear_smbus_cntr = 0;
//
// ---------------------------------------------------------------------
if(ad_counter == 0)                  // is it time to measure thermistor ?
{                                    // checks UV (DCOK input) also
  ad_th();                           // if ad_counter starts at 5, td=140ms
ad_counter = 5;
}
ad_counter--;
// ---------------------------------------------------------------------
if(flag.0)                           // retrun from I2C interrupt ?

{
    PORTB.3 = 0;                     // red LED on
    delay();                         // 1 second delay
    PORTB.3 = 1;                     // red LED off
    flag.0 = 0;
    i2c_counter = 0;                 // reset I2C stack pointer
    // com_timeout_cntr = 0;
    flag.1 = 0;                      // change to slow blinking
}
// ---------------------------------------------------------------------
if(i2c_data[1] == 0x16)              // AlarmWarning() on stack ?
{
   if( i2c_data[3] > 0x0f)
   {
     PORTC.0 = 1;
     flag.1 = 1;                     // set fast blinking
     i2c_data [1] = 0xF6;            // erase command from stack
     PORTC.0 = 1;                    // shutdown = 1 (inhibit charger)
     com_timeout_cntr = 0;
     flag.3 = 0;                     // enable com_timeout_cntr
   }
}
// ---------------------------------------------------------------------
if(i2c_data[1] == 0x14)              // ChargingCurrent() on stack ?
{
  load_ipwm();                       // set PWM
  i2c_data[1] = 0xF4;                // erase command
  flag.1 = 0;                        // set slow blinking
  com_timeout_cntr = 0;
  flag.3 = 0;                        // enable com_timeout_cntr
}
//---------------------------------------------------------------------
if(i2c_data[5] == 0x15)              // ChargingVoltage() on stack ?
{
  load_vpwm();                       // set output voltage
  i2c_data[5] = 0xF5;                // erase command
  flag.1 = 0;                        // set slow blinking
  PORTC.0 = 0;                       // shutdown = 0 enable charger
  com_timeout_cntr = 0;
  flag.3 = 0;                        // enable com_timeout_cntr
}
// -------------- setting LED blinking speed -------------------------
```


© 1997 Microchip Technology Inc.

```
    //
    if(flag.1)                          // fast blinking green requested ?
    {                                   // or shutdown
      LED_counter++;
      if(LED_counter > 1)               // is it time to change status of LED ?
      {
         LED_counter = 0;               // if yes, clear counter
         if(PORTB.2) PORTB.2=0;         // and toggle LED
         else PORTB.2 = 1;
      }
    }
    if(!flag.1)                         // slow flashing green - charging
    {
      LED_counter++;
      if(LED_counter > 4)               // time to change LED status ?
      {
         LED_counter = 0;               // if yes, reset counter
         if(PORTB.2) PORTB.2=0;         // and toggle LED
         else PORTB.2 = 1;
      }
    }
  } // end of while
}   // end of main
// -------------------------------------------------------------
//
void clear_smbus( void )
{
  TRISC.SDA = 1;                        // set as input
  TRISC.SCL = 1;                        // set as input
  //
  PORTC.SDA = 0;                        // to pull down SDA line when TRISC.SDA=L

  SSPCON.SSPEN = 0;                     // configure SDA and SCL pins as i/o pins
  //
     if( !PORTC.SCL) goto clsm1;        // SMBus traffic ? jump if yes
       delay_5us();
     if( !PORTC.SCL) goto clsm1;        // SMBus traffic ? jump if yes
       delay_5us();
     if( !PORTC.SCL) goto clsm1;        // SMBus traffic ? jump if yes
       //
  TRISC.SDA = 0;                        // SDA ->   ~~~\___    start
  delay_5us();
  TRISC.SDA = 1;                        // SDA ->   ___/~~~    stop
  //
  clsm1:
  PORTC.SDA = 1;
  TRISC.SDA = 1;                        // set as output
  TRISC.SCL = 1;                        // set as output
  SSPCON.SSPEN = 1;                     // cofigure SDA and SCL as serial port pins
}
//
//-------------------------------------------------------------
// initialize voltage and current PWMs.
//-------------------------------------------------------------
void initiv( char a )
{
  if( a==1)
  {
    i2c_data[1] = 0x14;                 // ChargingCurrent command
    i2c_data[2] = 0x64;                 // charging current L byte
    i2c_data[3] = 0x00;                 //                  H byte
    i2c_data[5] = 0x15;                 // ChargingVoltage command
    i2c_data[6] = 0xFF;                 // charging voltage L byte
    i2c_data[7] = 0xFF;                 //                  H byte
  }
  if( a==0)
```


© 1997 Microchip Technology Inc.

```
  {
    i2c_data[1] = 0x14;                 // Charging Current command
    i2c_data[2] = 0x32;                 // charging current L byte
    i2c_data[3] = 0x00;                 //                  H byte
    i2c_data[5] = 0x15;                 // Charging Voltage command
    i2c_data[6] = 0xFF;                 // charging voltage L byte
    i2c_data[7] = 0x8F;                 //                  H byte
  }
  load_ipwm();
  load_vpwm();
  //
  i2c_data[1] = 0;
  i2c_data[5] = 0;
}
//-------------------------------------------------------------
// Measures thermistor and controls the charger accordingly.
// ------------------------------------------------------------
void ad_th( void)
{
  char *ptr;
  char i,j;

  PORTA.5 = 1;                          // use 3.32k pull-up
  ad_val = 0;

  for( j=0; j<4; j++)
  {
    PIR1.ADIF = 0;
    ADCON0.GO = 1;
    while(ADCON0.GO);

    ad_val = ad_val + ADRES;
  }
  PORTA.5 = 0;                          // turn 3.32k pull-up off
  ad_val = ad_val / 4;
  //
  ptr = &ad_val;
  val = *ptr;
  //
  if( val > 220)                        // -- thermistor too cold---------
  {
     flag.1 = 1;
     flag.2 = 1;
     initiv(1);
  }

  if( (val < 221) && (val > 121) )      // thermistor in-range -----------
  {
     if(flag.2)                         // returns from thermistor error
     {
        initiv(1);                      // set trickle current
        PORTC.0 = 0;                    // start charger
        flag.2 = 0;                     // clear thermistor error flag
     }
  }

  if( (val < 122) && (val > 34) )       // thermistor too hot -----------
  {
     flag.1 = 1;                        // LED fast blinking
     flag.2 = 1;                        // set thermistor  flag
     initiv(1);                         // set trickle charge
  }

  if( (val<35) && (val > 10) )          // Li ION
  {
    if(flag.2)
```

```
    {
       initiv(0);
       PORTC.0 = 0;                    // start charger
       flag.2 = 0;                     // clear error flag
    }
  }
  if( val< 11)                         // Thermistor shorted
  {
     flag.1 = 1;                       // green LED fast blinking
     flag.2 = 1;                       // set thermistor flag
     initiv(1);                        // set trickle charge
  }
}
//-------------------------------------------------------------
// reads current values from i2c_dat[2] and i2c_data[3] locations,
// (L and H bytes) limits  the current, scales current word, and
// loadx PWM registers (10 bit mode)
//-------------------------------------------------------------------
void load_ipwm( void )
{
  long idata;
  char *ptr;
  bits ilowbits;
  //
  ptr = &idata;                        // get address of idata
  //
  *ptr = i2c_data[2];                  // load L byte of idata
  *(ptr+1) = i2c_data[3];              // load H byte of idata
  //
  if(idata > 2600) idata = 2600;
  //
  idata >>=2;                          // scale idata 4096mA / 1024 = 4
  ilowbits = *ptr;                     // save L byte
  //
  idata >>=2;                          // upper 8 bit of data
  //
  CCP2CON.CCP2X = ilowbits.1;          // load lower two LSBits of 10 bit word to PWM
  CCP2CON.CCP2Y = ilowbits.0;
  CCPR2L = *ptr;                       // load upper 8 bit of 10 bit PWM word
}
//
//-------------------------------------------------------------------
// Reads voltage data from i2c_data[6] (L byte) and i2c_data[7] locations,
// assembles a 16 bit word. Limits the voltage at max 20V, scales vdata
// voltage data word and loads PWM with 10 bit data.
//-------------------------------------------------------------------------
void load_vpwm( void )
{
  long unsigned vdata;
  char *ptr;
  bits vlowbits;
  //
  ptr = &vdata;                        // get address of idata
  *ptr = i2c_data[6];                  // load L byte of idata
  *(ptr+1) = i2c_data[7];              // load H byte of idata
  //
  if(vdata < 8000) vdata = 8000;       // PWM1 0x000 =8V,  0x3FF = 20V
  if(vdata > 18000) vdata=18000;       // limit incoming voltage to 20V
  //
  vdata = vdata-6000;                  //
  //
  vdata >>= 4;                         // scale vdata 22-6=16V,  16V/1024=16
  vlowbits = *ptr;                     // save L byte
  vdata >>=2;                          // upper 8 bit of data
  //
  CCP1CON.CCP1X = vlowbits.1;          // load lower two LSBits of 10 bit word
```


© 1997 Microchip Technology Inc.

```
  CCP1CON.CCP1Y = vlowbits.0;        // to PWM 9-th and 10-th bits.
  CCPR1L = *ptr;                     // load upper 8 bit of 10 bit PWM word
}
//---------------------------------------------------------------------------
void delay(void)
{
  char i, j, k;                      // software delay, about 1 sec
  for(k=0; k<2; k++)                 // measured: 840 msec.
  {
    for(j=0; j<255; j++)
    {
      for(i=0; i<255; i++);          // inner loop delay: 1.647ms
    }
  }
}
// ---------------------------------------------------------------------
void delay1( void )
{
  char i,j;
  for(j=0; j<20; j++)
  {
    for(i=0; i<155; i++);            // delay 1ms
  }
}
// ---------------------------------------------------------------------
void delay_5us( void )             // 5us delay
{
#asm
  nop
  nop
  nop
  nop
#endasm
}
//---------------------------------------------------------------------------
void timer0( char a )              // td=(256-a)*256us @ 4MHz (prescaler /256)
{
  TMR0 = a;                        // reload load timer
  INTCON.T0IF = 0;                 // reset TMR0 interrupt flag
}
//---------------------------------------------------------------------------
//
void init_var( void )
{
//------INIT TMR0--------------
//
  OPTION = 0b11010111;// init timer 1, no pull-up at b (bit7=1)
                                   // bit7 Port B pull-up enable (1=disable)
                                   //      INTEDG 1=INT on rising of RB0/INT
                                   //      T0CS TMR0 ck source 0 = internal
                                   //      T0SE TMR0 source edge 1=H->L on RA4
                                   // bit3 PSA prescaler assign. 1=WDT, 0=TMR0
                                   //      PS2:PS0 prascaler div. rate 2,4,8,16..

//--- TMR2 and PWMs ----------
//
  T2CON = 0x04;// init timer, bit2 turns it on
                                   // bit7 unimplemented
                                   // bit6:bit3 postscaler select 1,2,3..16
                                   // bit2 TMR2ON 1=TMR2 on 0=TMR2 off
                                   // bit1:bit0 prescaler div. 1, 4 or 16
//
  CCP1CON = 0b00001100;// init PWM1
                                   // bit7 unimplemented
                                   //      unimplemented
                                   //      bit1 for 10 bit mode (0 for 8bit)
```


© 1997 Microchip Technology Inc.

```
                                //      bit0 for 10 bit mode (0 for 8bit)
                                // bit3:0 mode select 11xx= PWM mode
//
  CCP2CON = 0b00001100;// init PWM2

// --- init port A --
  PORTA = 0;
  //        76543210
  TRISA = 0b11011011;           // RA5  3.32k       OUT
                                // RA4  header 4    IN
                                // RA3  header 3    IN
                                // RA2  33.2k       OUT
                                // RA1  UV          IN
                                // RA0  analog in   IN
// ----init A/D------
  //         76543210
  ADCON0 = 0b01000001;          // Analog digital converter module
                                // bit7 ADCS1   A/D clock select
                                //      ADCS0   01= fosc/8 -> tconv=16us
                                //      CHS2    channel selection
                                //      CHS1
                                // bit3 CHS0    000 -> RA0
                                //      GO/DONE_ start conv/finished
                                //      unimplemented
                                //      ADON 1= a/d on 0=a/d off
  //         76543210
  ADCON1 = 0b00000100;          // b7:b3 not implemented
                                // b2:b0 analog port pin config.
                                // 100   RA0=analog  RA1=analog
                                //       RA2=digital RA3=digital, Vref=VDD

// ----init port B---
//
  PORTB  = 12;
  //        76543210
  TRISB = 0b11110011;           // RB7 (pin 28) -> header 11    IN
                                // RB6 (pin 27) -> header 10    IN
                                // RB5 (pin 26) DCOK input      IN
                                // RB4 (pin 25) -> header 5     IN
                                // RB3 red  LED                 OUT
                                // RB2 grn. LED                 OUT
                                // RB1 (pin 22) -> header 9     IN
                                // RB0 (pin 21) -> header 8     IN
// ----init.port C---

  PORTC = 1;
  //        76543210
  TRISC = 0b11011000;// init PORTC  C.2 = CCP1 C.1 = CCP2
                                // RC7 (pin18) header 7
                                // RC6 (pin17) header 6
                                // RC5 (pin16) header 5
                                // RC4 (pin15) SDA  I2C data  -> input
                                // RC3 (pin14) SCL  I2C clock -> input
                                // RC2 (pin13) CCP1 PWM1 pin, -> output
                                // RC1 (pin12) CCP2 PWM2 pin, -> output
                                // RC0 (pin11) shutdown       -> output

  // -------------------------
  CCPR1L = 10;                  // pulse width1
  CCPR2L = 10;                  // pulse width2
  PR2 = 255;                    // period time 200-> 200us

  //---setting-up I2C communication ----------
  //         76543210
  SSPCON = 0b00110110;          // sync serial port control register
                                // B7 WCOL=0    Write collision det. (SW reset)
```


© 1997 Microchip Technology Inc.

```
                                //    SSPOV=0   receive collision det (SW reset)
                                //    SSPEN=1   enable ser port ( SCK SDO open D)
                                //    CKP = 1   0=enable clock
                                // B2:B0 SSPM2:SSPM0=110 slave mode.
// --- timer interrupts---
  INTCON.T0IF = 1;              // reset TMR0 int flag
  INTCON.T0IE = 0;              // 1=enable TMR0 interrupt

  PIE1.SSPIE = 1;               // enable i2c interrupt
  PIR1.SSPIF = 0;               // reset i2c interrupt flag

//  T1CON = 0x00;               // init TMR1, internal ck, prescaler div=1

  i2c_data[0] = 0;              // clear address locations
  i2c_data[1] = 0;
  i2c_data[2] = 0;
  i2c_data[3] = 0;
  i2c_data[4] = 0;
  i2c_data[5] = 0;
  i2c_data[6] = 0;
  i2c_data[7] = 0;
  i2c_data[8] = 0;
  i2c_data[9] = 0;
}
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

SECTION 4
PIC17CXXX
APPLICATION NOTES

4

Servo Control of a DC-Brush Motor - AN532 4-1
Saving and Restoring Status on Interrupt (Implementing a Parameter Stack) - AN534 4-141
Using PWM to Generate Analog Output - AN538 4-143
Frequency and Resolution Options for PWM Outputs - AN539 4-145
Implementing IIR Digital Filters - AN540 4-157
Implementation of Fast Fourier Transforms - AN542 4-177
Tone Generation - AN543 4-199
Math Utility Routines - AN544 4-209
Using the Capture Module - AN545 4-259
Serial Port Utilities - AN547 4-283
Implementing Table Read and Table Write - AN548 4-295
Using the PWM - AN564 4-299
Implementation of the Data Encryption Standard Using PIC17C42 - AN583 4-311

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

MICROCHIP

AN532

Servo Control of a DC-Brush Motor

Author: Tim Bucella
Teknic, Inc.

INTRODUCTION

The PIC17C42 microcontroller is an excellent choice for cost-effective servo control in embedded applications. Due to its Harvard architecture and RISC features, the PIC17C42 offers excellent computation speed needed for real-time closed loop servo control. This application note examines the use of the PIC17C42 as a DC brush motor servo controller. It is shown that a PID (Proportional, Integral, Differential) control calculation can be performed in less than 200 µs (@16 MHz) allowing control loop sample times in the 2 kHz range. Encoder rates up to 3 MHz are easily handled by the PIC17C42's high speed peripherals. Further, the on-chip peripherals allow an absolute minimum cost system to be constructed.

Closed-loop servo motor control is usually handled by 16-bit, high-end microcontrollers and external logic. In an attempt to increase performance many applications are upgrading to DSPs (Digital Signal Processors). However, the very high performance of the PIC17C42 makes it possible to implement these servo control applications at a significant reduction in overall system cost.

The servo system discussed in this application note uses a PIC17C42 microcontroller, a programmable logic device (PLD), and a single-chip H-bridge driver. Such a system might be used as a positioning controller in a printer, plotter, or scanner. The low cost of implementing a servo control system using the PIC17C42 allows this system to compete favorably with stepper motor systems by offering a number of advantages:

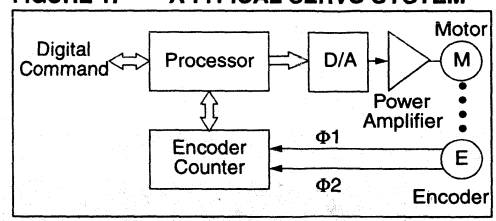
- Increased Acceleration, Velocity
- Improved Efficiency
- Reduced Audible Noise
- True Disturbance Rejection

SYSTEM OVERVIEW

DC Servo Control

Modern digital servo systems are formed as shown in Figure 1. These systems control a motor with an incremental feedback device known as a sequential encoder. They consist of an encoder counter, a processor, some form of D/A (Digital-to-Analog) converter, and a power amplifier which delivers current or voltage to the motor.

FIGURE 1: A TYPICAL SERVO SYSTEM



The PIC17C42 implements both the servo compensator algorithm and the trajectory profile (trapezoidal) generation. A trajectory generation algorithm is necessary for optimum motion and its implementation is as important as the servo compensator itself. The servo compensator can be implemented as a traditional digital filter, a fuzzy logic algorithm, or a simple PID algorithm (as implemented in this application note). The combination of servo compensator and trajectory calculations can place significant demands on the processor.

The D/A conversion can be handled by a conventional DAC or by using the PIC17C42's pulse-width modulation (PWM). In either case the output signal is fed to a power stage which translates the analog signal(s) into usable voltages and currents to drive the motor.

PWM output can be a duty-cycle signal in combination with a direction signal or a single signal which carries both pieces of information. In the latter case a 50% duty cycle commands a null output, a 0% duty cycle commands maximum negative output, and 100% maximum positive output.

The amplifier can be configured to supply a controlled voltage or current to the motor. Most embedded systems use voltage output because its simpler and cheaper.

Sequential encoders produce quadrature pulse trains, from which position, speed, and direction of the motor rotation can be derived. The frequency is proportional to speed and each transition of F1 and F2 represents an increment of position. The phase of the signals is used to determine direction of rotation.

These encoder signals are usually decoded into Count Up and Count Down pulses, using a small state machine. These pulses are then routed to an N-bit, up/down counter whose value corresponds to the position of the motor shaft. The decoder/counter may be implemented in hardware, software, or a combination of the two.

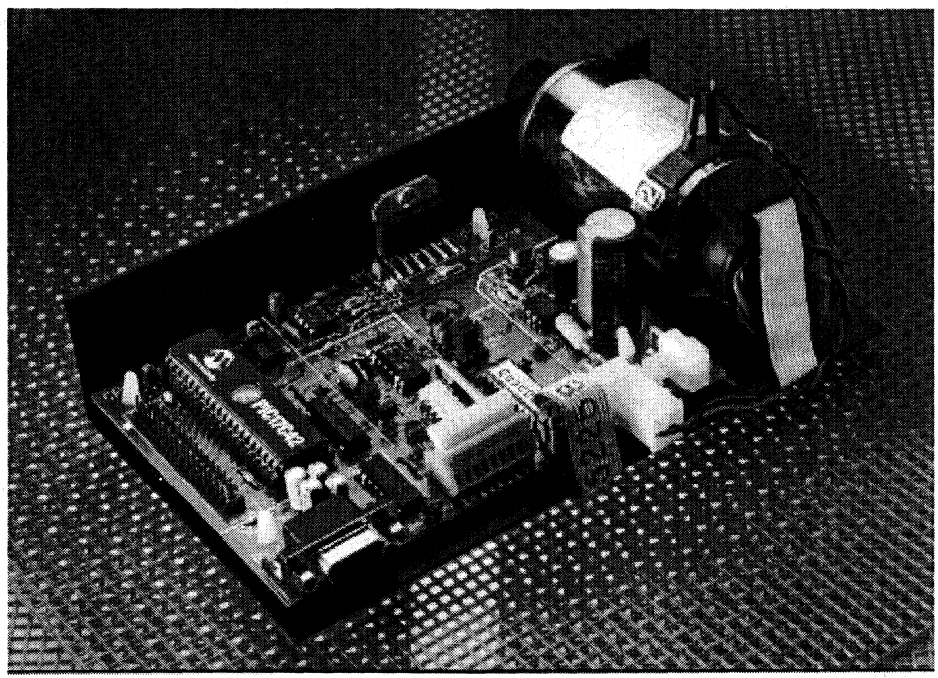
4

© 1997 Microchip Technology Inc.

The PIC17C42 Based Motor Control Board

The PIC17C42 based servo system described here has a full RS-232 ASCII interface, on-board switching power supply, H-bridge motor drive, over-current protection, limit switch inputs and digital I/O. The entire system measures 5" x 3.5" and is shown in Figure 2. The system can be used to evaluate the PIC17C42 in servo applications. All unused PIC17C42 pins are available at an I/O connector for prototyping.

FIGURE 2: THE PIC17C42 BASED SERVO CONTROL BOARD



© 1997 Microchip Technology Inc.

A PID algorithm is used as a servo compensator and position trajectories are derived from linear velocity ramp segments. This system uses 50%-null PWM as the D/A conversion technique. The power stage is a high current output switching stage which steps-up the level of the PWM signal. Encoder signal decoding is accomplished using an external PLD. The up/down counter is implemented internally in the PIC17C42 as combination of hardware and software (Figure 3 and Figure 4).

FIGURE 3: SEQUENTIAL ENCODER SIGNALS

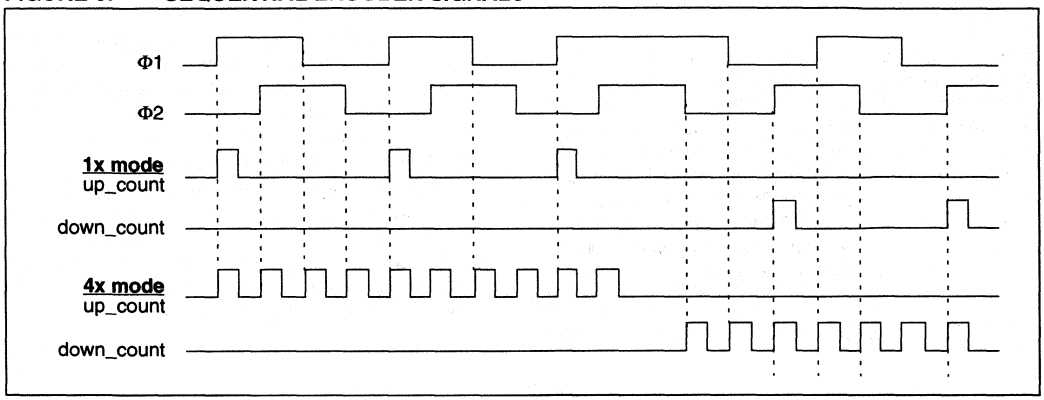
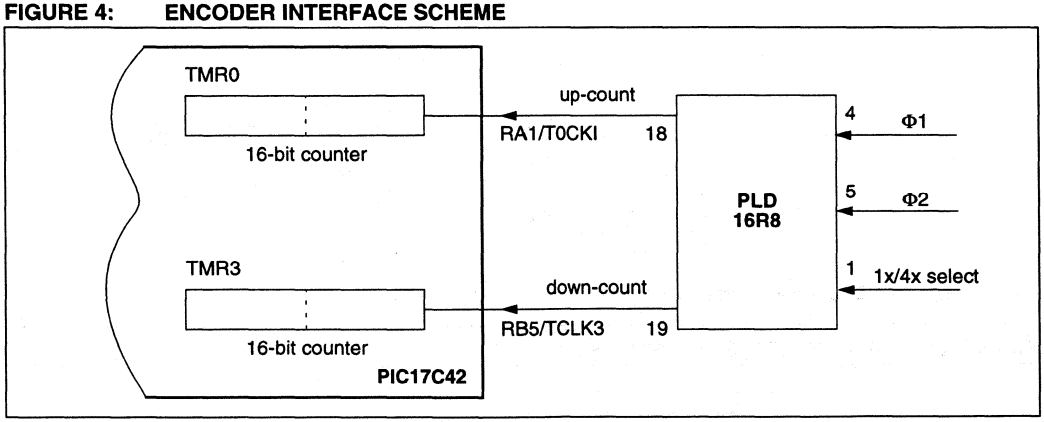


FIGURE 4: ENCODER INTERFACE SCHEME



THE COMPENSATOR

A PID routine is the most widely used algorithm for servo motor control. Although it may not be the most optimum controller for all applications, it is easy to understand and tune.

The standard digital PID algorithm's form is shown in Figure 5. *U(k)* is the position or velocity error and *Y(k)* is the output.

This algorithm has been implemented using the PIC17C42's math library. Only 800 instruction cycles are required, resulting in a 0.2 ms PID execution time at 16 MHz.

Integrator windup is a condition which occurs in PID controllers when a large following error is present in the system, for instance when a large step disturbance is encountered. The integrator continually builds up during this following error condition even though the output is saturated. The integrator then "unwinds" when the servo system reaches its final destination causing excessive oscillation. The PID implementation shown in Figure 5 avoids this problem by stopping the action of the integrator during output saturation.

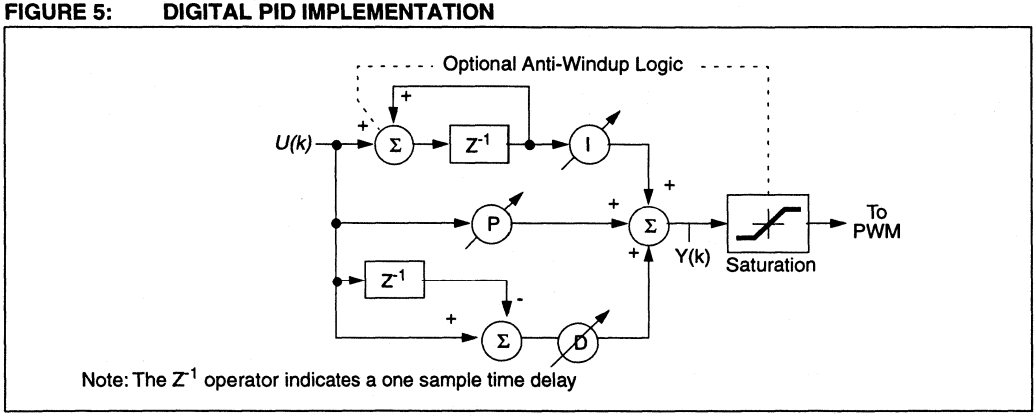
MOTOR ACTUATION

The PIC17C42 contains a high-resolution pulse width modulation (PWM) subsystem. This forms a very efficient power D/A converter when coupled to a simple switching power stage. The resolution of the PIC17C42 PWM subsystem is 62.5 ns (at 16 MHz). This translates into 10-bit resolution at a 15.6 kHz rate or 1 part in 800 (9 1/2-bit) resolution at 20 kHz. This allows effective voltage control while still maintaining the modulation frequency at or above the limit of human hearing. This is especially relevant in office automation equipment where minimizing noise is a design goal.

The motor responds to a PWM output stage by time averaging the duty cycle of the output. Most motors react slowly, having an electrical time constant of 0.5 ms or more and a mechanical time constant of 20.0 ms or more. A 15 kHz PWM output is effectively equivalent to that of a linear amplifier.

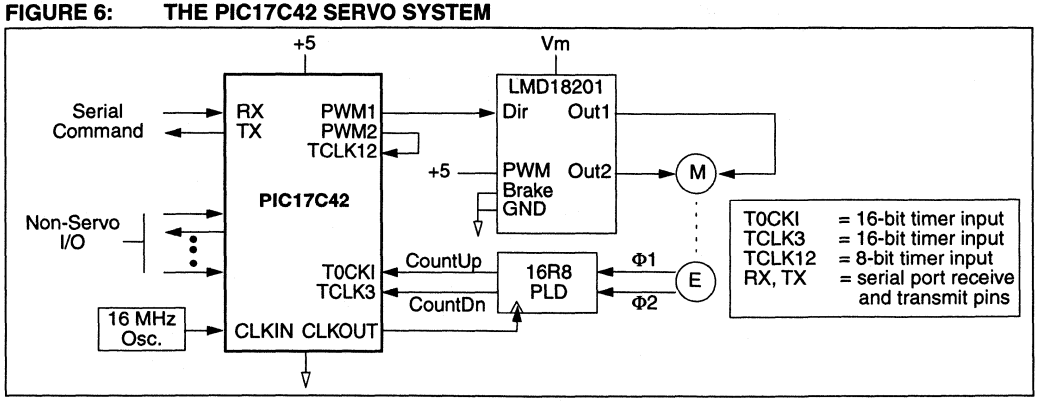
In the system shown in Figure 6, the H-bridge's direction input is wired directly to the PIC17C42's PWM output. The H-bridge is powered by a DC supply voltage, V_m. In this configuration 0 volts is presented to the motor when the PWM signal is at a 50% duty cycle, $-V_m$ volts at 0% duty cycle and $+V_m$ volts at 100% duty cycle.

FIGURE 5: DIGITAL PID IMPLEMENTATION



Note: The Z^{-1} operator indicates a one sample time delay

FIGURE 6: THE PIC17C42 SERVO SYSTEM



© 1997 Microchip Technology Inc.

ENCODER FEEDBACK

Position feedback for the example system is derived from a quadrature encoder mounted on the motor shaft. Both incremental position and direction can be derived from this inexpensive device. The quadrature encoder signals are processed by a 16R8-type PLD device as shown in Figure 6. The PLD converts the quadrature pulses into two pulse streams: Count Up and Count Down (Figure 3). These signals are then fed to two 16-bit timers of the PIC17C42 (Timer3 and Timer0). A logic description for the PLD decoder is shown in Appendix B.

The PIC17C42 keeps track of the motor shaft's incremental position by differencing these two 16-bit timers. This operation is performed each servo sample time and the current position is calculated by adding the incremental position to the previous position. Since both timers are 16-bits, keeping track of the overflow is unnecessary, unless the encoder signals frequency is greater than 32767 times the sample frequency. **For example, at a servo sample time of 1 ms, the maximum encoder rate would be 3.2767 MHz.**

Counter wraparound is not a concern because only the difference between the two counters is used. Two's-complement subtraction takes care of this automatically. Position is maintained as a three-byte, 24-bit quantity in the example program shown in Appendix F. However, there is no limit to the size of the internal position register. By adding the 16-bit incremental position each sample time to an N-byte software register, an N-byte position may be maintained.

TRAJECTORY GENERATION

A trajectory generation algorithm is essential for optimum motion control. A linear piecewise velocity trajectory is implemented in this application. For a position move, the velocity is incremented by a constant acceleration value until a specified maximum velocity is reached. The maximum velocity is maintained for a required amount of time and then decremented by the same acceleration (deceleration) value until zero velocity is attained. The velocity trajectory is therefore trapezoidal for a long move and triangular for a short move where maximum velocity was not reached (Figure 7).

The `doPreMove` subroutine is invoked once at the beginning of a move to calculate the trajectory limits. The `doMove` routine is then invoked at every sample time to calculate new "desired" velocity and position values as follows:

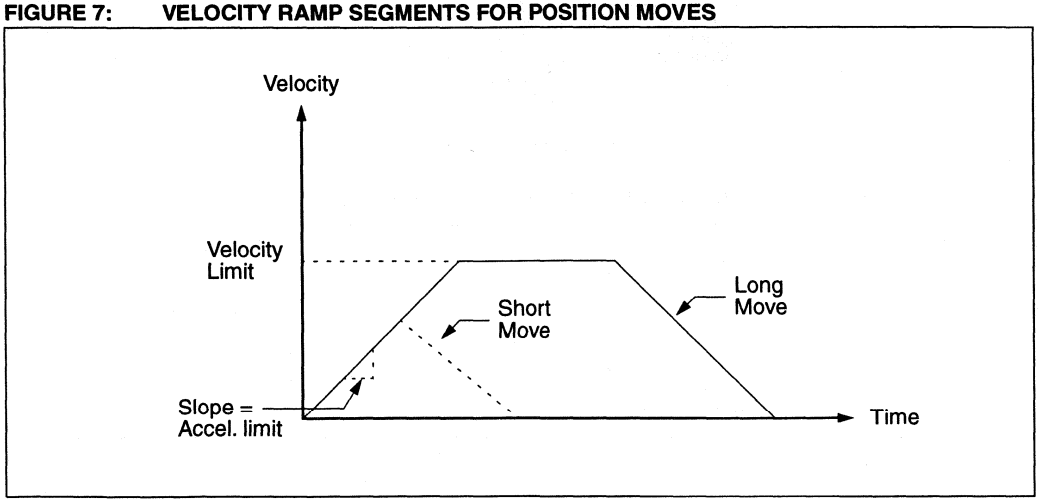
$$VK = VK\text{-}1 + A \quad (A = \text{Acceleration})$$

$$PK = PK\text{-}1 + VK\text{-}1 + A/2$$

For more details on trajectory generation, see Appendix E.

4

FIGURE 7: VELOCITY RAMP SEGMENTS FOR POSITION MOVES

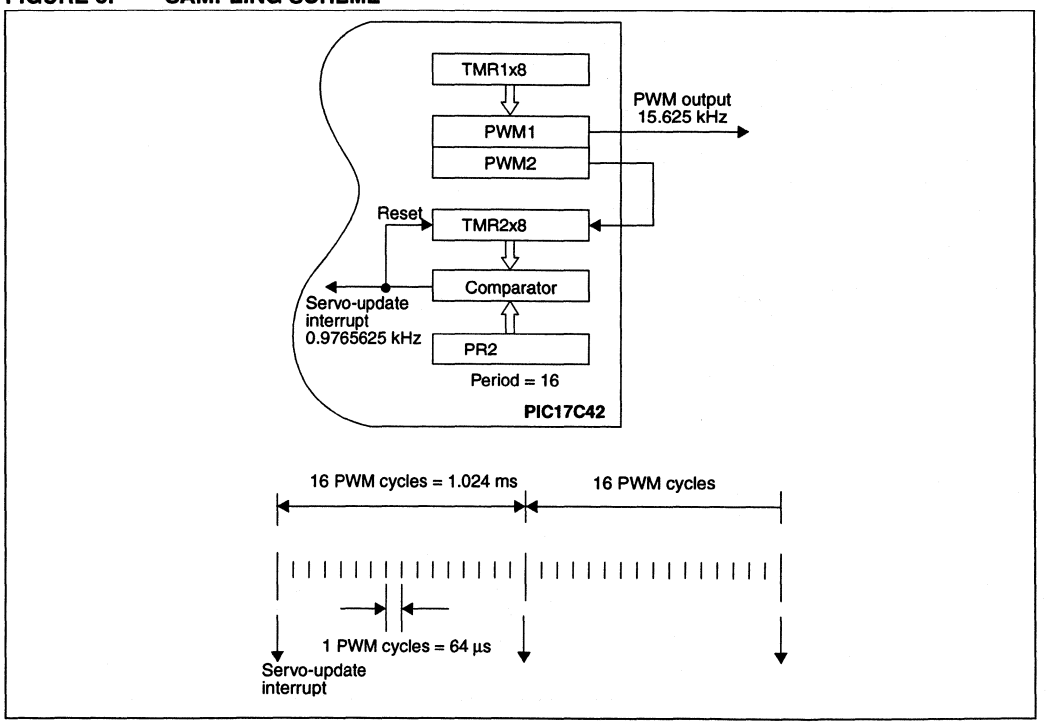


IMPLEMENTATION DETAILS

The program structure is straightforward: An interrupt service routine (ISR) processes the servo control and trajectory generation calculations, and a foreground loop is used to implement the user interface, serial communication, and any exception processing (i.e., limit switches, watchdog timer, etc.).

The ISR has a simple structure. In order to effect servo control we need to read the encoder, calculate the new trajectory point and PID values, and set the output of the PWM, all at a constant, predefined rate. The ISR is initiated by a hardware timer (Timer2) on the PIC17C42. To make sure that the servo calculation always occurs synchronously with the PWM subsystem, the PWM2 output is wired to the input pin of TMR12 (TMR1 in internally-clocked, 8-bit timer mode; TMR2 in externally-clocked, 8-bit counter mode). N is loaded into the PR2 register. The sample rate then becomes the PWM rate divided by N. In this implementation N = 16 (Figure 8).

FIGURE 8: SAMPLING SCHEME



© 1997 Microchip Technology Inc.

FIGURE 9: FLOWCHART FOR FOREGROUND PROCESSING

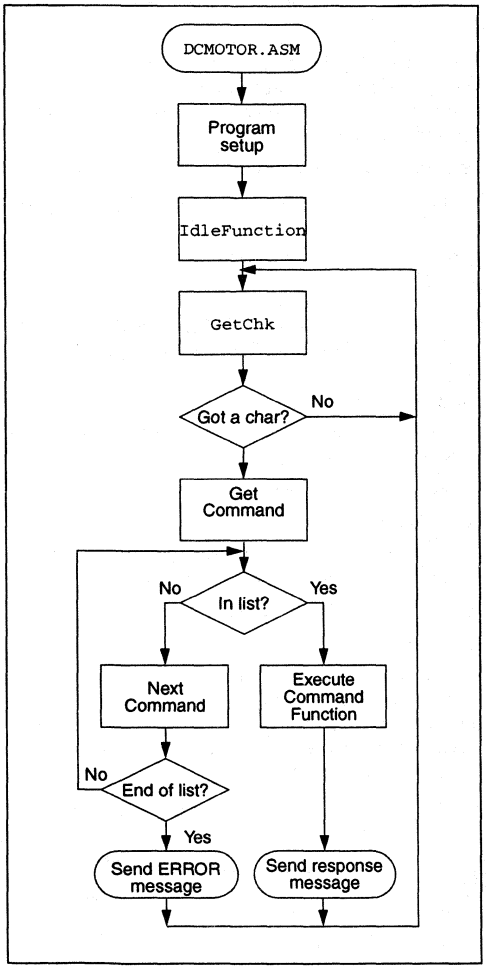
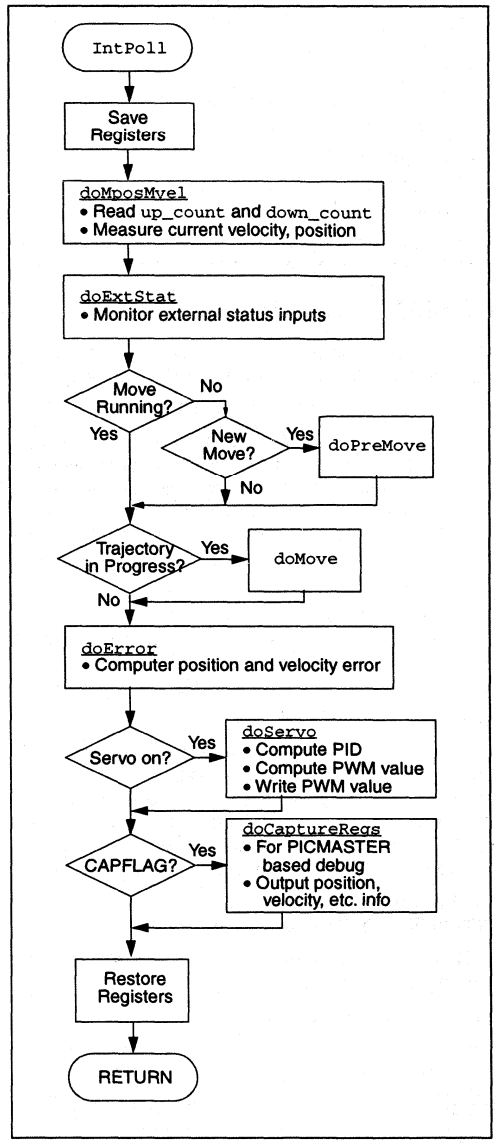


FIGURE 10: FLOWCHART FOR INTERRUPT SERVICE ROUTINE



4

The following events must occur in the interrupt service routine:

- Read Timers (TMR0 & TMR3)
- Calculate the new Reference Position using the Trajectory Generation Routine.
- Calculate Error:
  U(k) = Reference Position - Current Position
- Calculate Y(k) using PID
- Set PWM output
- Manage other housekeeping tasks
  (i.e. service serial characters)

The entire ISR requires only 0.250 ms to execute with 16 MHz processor clock frequency.

COMMAND INTERFACE

The following commands are implemented and recognized by the user interface in the foreground loop.

Move (Value): M, [$-8,388,608_{10}$ to $8,388,607_{10}$]

Commands the axis to move to a new position or velocity. Position data is relative, velocity data is absolute. Position data is in encoder counts. Velocity data is given in encoder counts per sample time multiplied by 256. All moves are performed by the controller such that velocity and acceleration limits set into parameter memory will not be violated.

All move commands are kept in a one deep FIFO buffer. The command in the buffer is executed as soon as the executing command is complete. If no move is currently executing the commanded move will start immediately.

Mode: O, (Type), [P,V, T]

An argument of "P" will cause all subsequent move commands to be incremental position moves. A "V" argument will cause all subsequent moves to be absolute velocity moves. A "T" argument sets a "Torque mode'" where all subsequent M commands directly write to the PWM. This is useful for debug purposes.

Set Parameter: S, (#,Value)
[00h to FFh, $-8,388,608_{10}$ to $8,388,607_{10}$]

Sets controller parameters to the value given. Parameters are shown in Table 1.

TABLE 1: PARAMETERS

Parameter	#	Range
Velocity Limit	00h	0 to $8,388,607_{10}$ *
Acceleration Limit	01h	0 to $8,388,607_{10}$ **
Kp: Proportional Gain	02h	-32768_{10} to 32767_{10}
Kd: Differential Gain	03h	-32768_{10} to 32767_{10}
Ki: Integral Gain	04h	-32768_{10} to 32767_{10}

* (counts per sample time multiplied by 256)
** (counts per sample time per sample time multiplied by 256)

Read Parameter: R, (#) [00h to FFh]

Returns the present value of a parameter.

Shutter: C

Returns the time (in sample time counts 0 to $65,536_{10}$) since the start of the present move and captures the commanded and actual values of position and velocity at the time of the command.

Read commanded position: P

Returns the commanded position count which was captured during the last Shutter command.

Range: $-8,388,608_{10}$ to $8,388,607_{10}$.

Read commanded velocity: V

Returns the commanded velocity multiplied by 256 which was captured during the last Shutter command. Range: $-8,388,608_{10}$ to $8,388,607_{10}$.

Read actual position: p

Returns the actual position count which was captured during the last Shutter command.

Range: $-8,388,608_{10}$ to $8,388,607_{10}$.

Read actual velocity: v

Returns the actual velocity multiplied by 256 which was captured during the last Shutter command.

Range: $-8,388,608_{10}$ to $8,388,607_{10}$.

External Status:

Returns a two digit hex number which defines the state of the bits in the external status register. Issuing this command will clear all the bits in the external status register unless the event which set the bit is still true. The bits are defined in Table 2.

TABLE 2: EXTERNAL STATUS REGISTER BITS

bit 7	index marker detected
bit 6	+limit reached
bit 5	-limit reached
bit 4	input true
bit 3-0	N/A

Move Status: Y

Returns a two-digit hex number which defines the state of the bits in the move status register. Issuing this command will clear all the bits in the move status register unless the event which set the bit is still true. The bits are defined in Table 3.

TABLE 3: MOVE STATUS REGISTER BITS

bit 7	move buffer empty
bit 6	move complete
bit 5-0	N/A

© 1997 Microchip Technology Inc.

Read Index position: I

Returns the last index position captured in position counts.

Set Position (Value): H, [-8,388,60810to8,388,60710]

Sets the actual and commanded positions to the value given. Should not be sent unless the move FIFO buffer is empty.

Reset: Z

Performs a software reset.

Capture Servo-Response: c (#Count)

The c command will set a flag inside indicating that starting with the next M (servo move) command, velocity and position information will be sent out (by invoking the `doCaptureRegs` procedure) during every servo-loop for #count times. At the end of the #count, the processor will halt (see `doCaptureRegs` procedure). This is useful for debug purposes.

Disable Servo: s

This command disables servo actuation. The servo will activate again with the execution of the next M (move) command. This is useful for debug purposes.

Examples:

```
Z           ;Reset software (No <CR> required)
OV          ;Set velocity servo mode
            ;(No <CR> required)
M 1000<CR>  ;Set velocity to 1000
M-1000<CR>  ;Set velocity to 1000 in reverse
            ;direction
```

OPTIMIZING THE SYSTEM

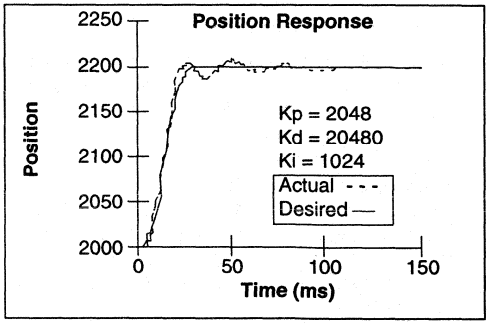
Once the PID loop is successfully implemented, the next challenge is to tune it. This was made simple through extensive use of the PICMASTER™ In-Circuit Emulator for the PIC17C42.

The PICMASTER is a highly sophisticated real-time in-circuit emulator with unlimited break-point capability, an 8K deep trace buffer and external logic probes. Its user interface software runs under Windows® 3.1 with pull-down menus and on-line help. The PICMASTER software also supports dynamic data exchange (DDE). The DDE makes it possible to send its trace buffer information to a spreadsheet, such as EXCEL®, also running under Windows.

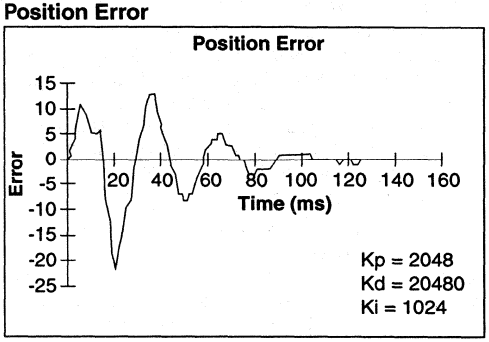
To tune the PID, first a small amount of diagnostics code is added in the servo routine (`doCaptureRegs`). This code simply outputs, at every sample point, the actual and desired position values, actual and desired velocity values, position error and velocity error by using a `TABLWT` instruction. These are captured in the trace buffer of the emulator. The 'trace' condition is set up to only trace the data cycles of the 2-cycle `TABLWT` instructions. Next, the trace buffer is transferred to EXCEL and the various parameters are plotted. The plots graphically show the amounts of overshoot, ripple and response time. By altering Kp, Ki and Kd, and plotting the results, the system can be fine tuned.

FIGURE 11: TYPICAL SERVO RESPONSE

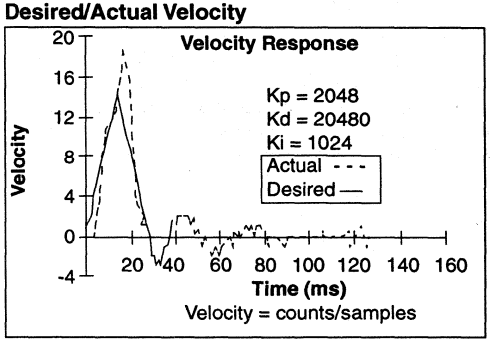
Desired/Actual Position



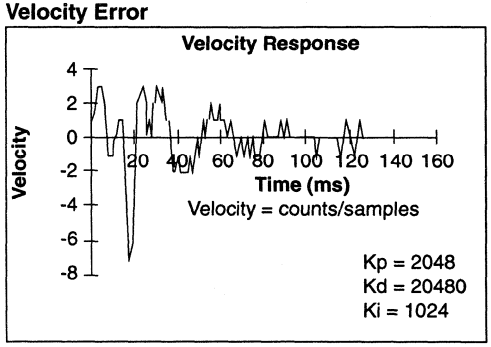
Position Error



Desired/Actual Velocity



Velocity Error



4

© 1997 Microchip Technology Inc.

Under Windows multi-tasking environment, using a PICMASTER emulator, this can be done in real time as described below.

Three sessions are set up under Windows:

1. A terminal emulator session to send commands to the motor control board. The "terminal" program provided with Windows is used, although any communications software such as PROCOMM will work.
2. Second, a PICMASTER emulation session is invoked. The actual PIC17C42 is replaced in-circuit by the emulator probe. Within the emulator, trace points are setup to capture the actual and desired position and velocity values on appropriate bus cycles.
3. Third, a session of EXCEL is started and dynamically linked to the PICMASTER sessions such that whenever the trace buffer is full, the data is sent over to EXCEL. A few simple filtering commands in EXCEL are used to separate the various data types, i.e. actual position data from desired position from actual velocity etc. Next, various plot windows are set up within EXCEL to plot these information.

Once these setups have been done, for every servo move, the responses are automatically plotted. It is then a simple matter of varying the PID coefficients and observing the responses to achieve the desired system response. At any point, the responses can be stored in files and/or printed out.

Except for very long "move" commands, most position and velocity commands are executed (i.e. system settled) in less than 500 samples, making it possible to capture all variables (actual and desired position and velocity, and position errors and servo output) in PICMASTER's 8K trace buffer.

CONCLUSIONS

Using a high-performance 8-bit microcontroller as the heart of a servo control system is a cost-effective solution which requires very few external components. A comparison with a popular dedicated servo-control chip, is presented in Table 4.

TABLE 4: SERVO CONTROL CHIP COMPARISON

	LM629 @8 MHz	PIC17C42 @16 MHz	PIC17C42 @25 MHz
Max Encoder Rate	1 MHz	3.3 MHz	4.5 MHz
Servo Update Time	-	0.25 ms	0.16 ms
Max Sampling Frequency	4 kHz	2-3 kHz	4-5 kHz

Also apparent in the comparison table is the additional processing power available when using the microcontroller. This processing can be used to provide a user interface, handle other I/O, etc. Alternatively, the additional processing time might be used to improve the performance of compensator and trajectory generation algorithms. A further advantage is that for many embedded applications using motor control the microcontroller proves to be a complete, minimum cost solution.

Credit

This application note and a working demo board has been developed by Teknic Inc. Teknic (Rochester, N.Y.) specializes in Motor Control Systems.

References

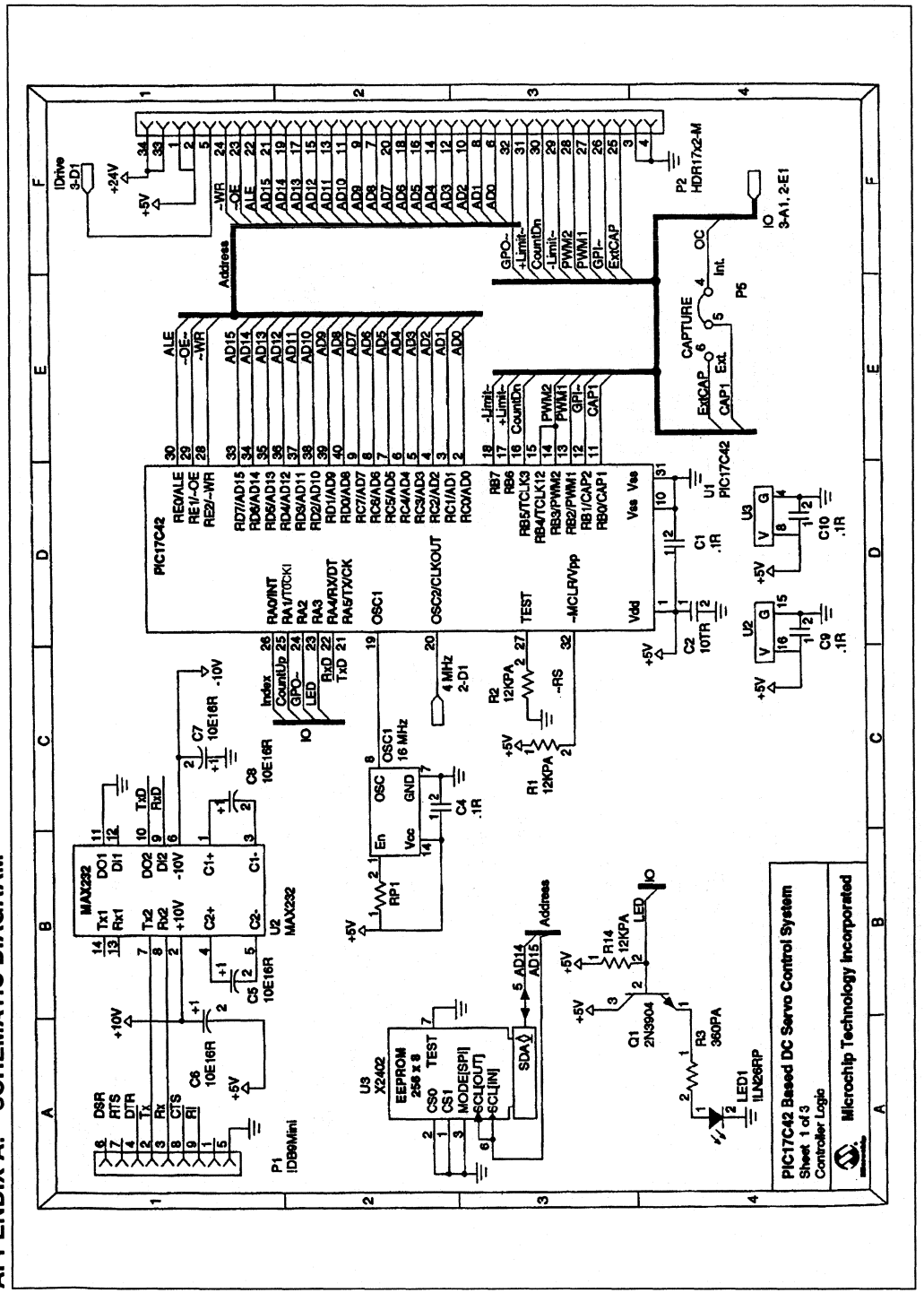
1.Thomas Bucella, "Comparing DSPs to Microprocessors in Motion Control Systems-Some Real World Data", PCIM conference proceedings © 1990 Intertec Communications, Inc.

2.David M. Auslander, Cheng H. Tham, "Real-Time Software for Control" © 1990 Prentice-Hall, Inc., Englewood Cliffs, NJ

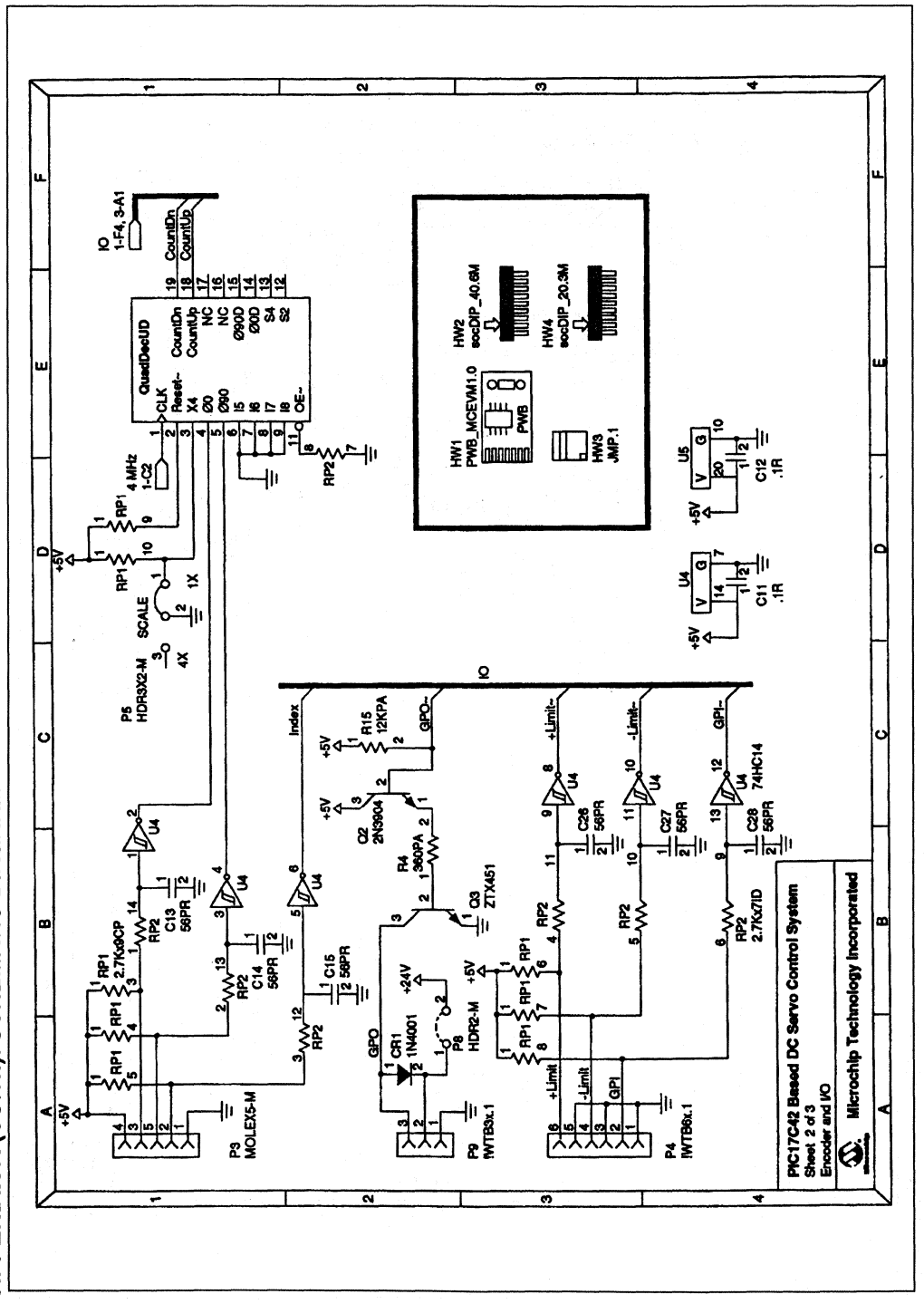
3."DC Motors, Speed Controls, Servo Systems" Fifth Edition © 1980 Electro-Craft Corporation, Hopkins, MN

© 1997 Microchip Technology Inc.

APPENDIX A: SCHEMATIC DIAGRAM

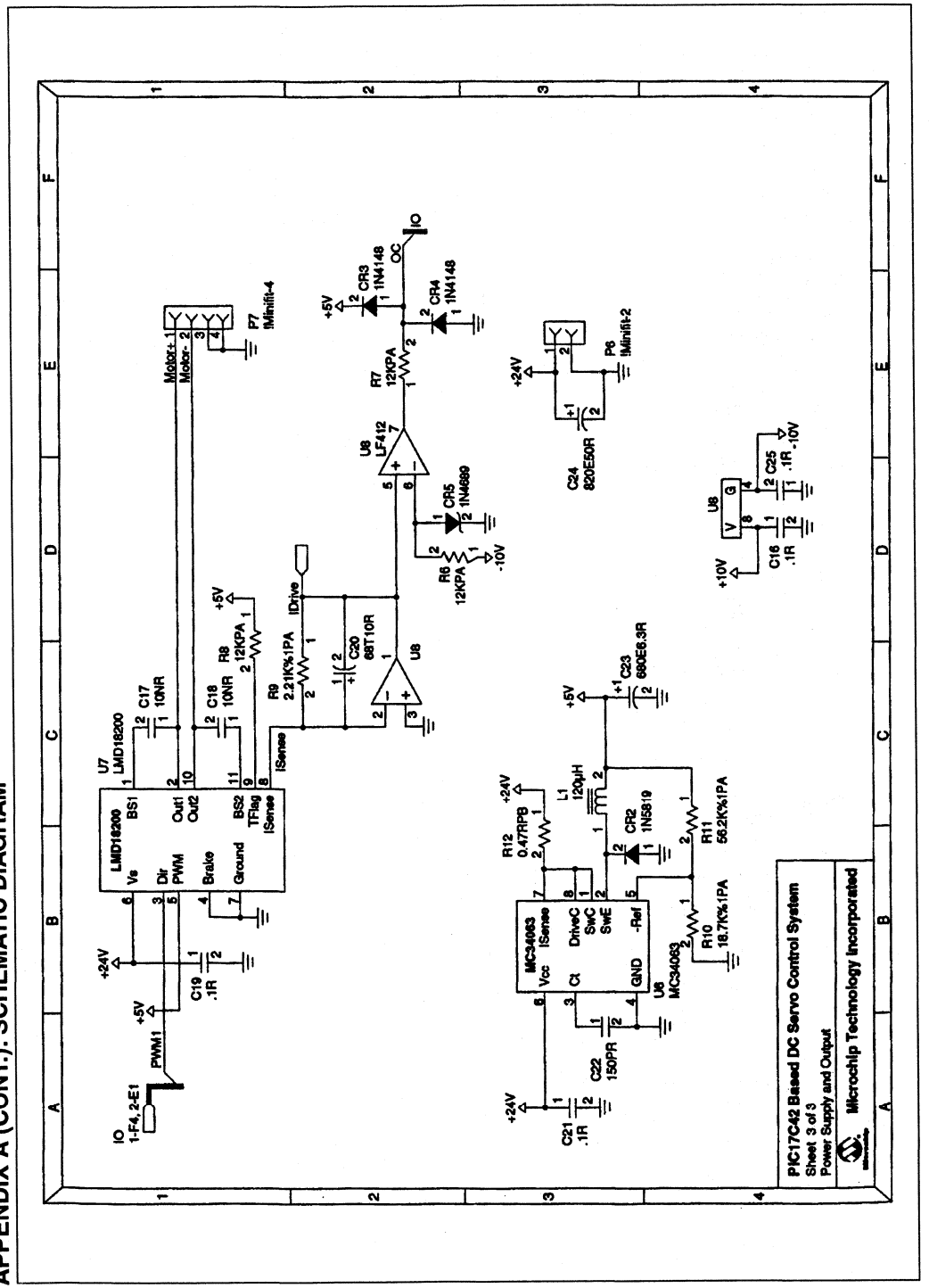


APPENDIX A (CONT.): SCHEMATIC DIAGRAM



© 1997 Microchip Technology Inc.

APPENDIX A (CONT.): SCHEMATIC DIAGRAM



© 1997 Microchip Technology Inc.

APPENDIX B:

```
Combination quadrature decoder and input synchronizer.  This design
allows 1x decoding or 4x decoding based on the X4 pin.

    * Ver 1.0 - November 8, 1991
}
MODULE QuadDivider;
TITLE QuadDivider V1.0;
COMMENT Device: 16R8;

TYPE MMI 16R8;
INPUTS;
        RESET NODE[PIN2] INVERTED;
        X4 NODE[PIN3];
            P0 NODE[PIN4];              { Phi0 }
        P90 NODE[PIN5];                 { Phi90 }
        INDX NODE[PIN6];
        { Feedback pins }
        S2 NODE[PIN12];
        S4 NODE[PIN13];
        P0D NODE[PIN14];
        P90D NODE[PIN15];
        CntUp NODE[PIN18];
        CntDn NODE[PIN19];
        UP NODE[PIN16];
        COUNT NODE[PIN17] INVERTED;
OUTPUTS;
        S2 NODE[PIN12];
        S4 NODE[PIN13];
        P0D NODE[PIN14];
        P90D NODE[PIN15];
        CntUp NODE[PIN18];
        CntDn NODE[PIN19];
        UP NODE[PIN16];
        COUNT NODE[PIN17] INVERTED;
TABLE;
        S2 := P0D & !RESET;
        S4 := P90D & !RESET;
        P0D := P0 & !RESET;
        P90D := P90 & !RESET;

        CntUp := COUNT & UP;
        CntDn := COUNT & !UP;

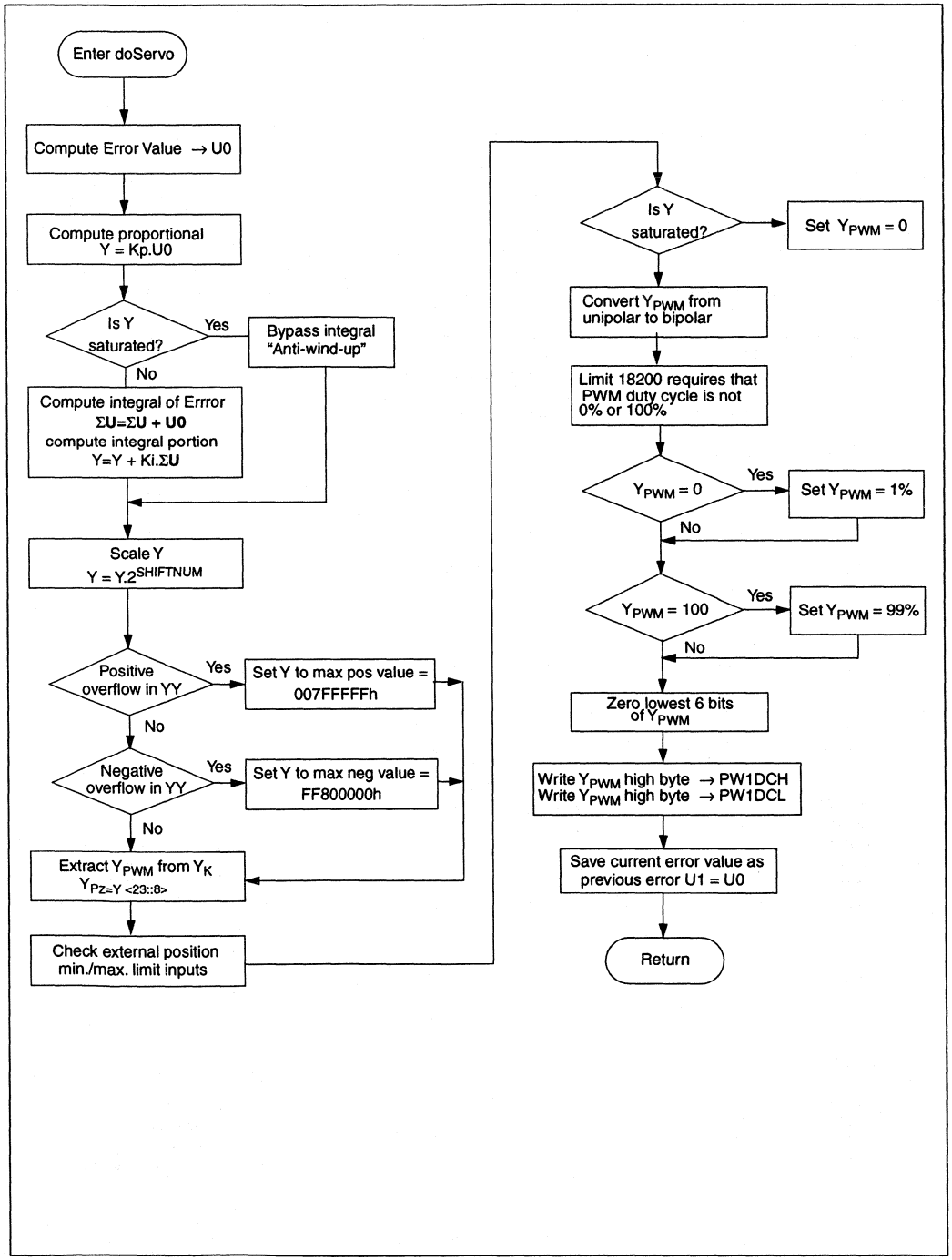
        COUNT :=
                (  P0D &  S2 & !P90D &  S4 & X4{ C1 }
                +!P0D & !S2 &  P90D & !S4     { C2 }
                +!P0D &  S2 & !P90D & !S4 & X4{ C3 }
                + P0D & !S2 &  P90D &  S4 & X4{ C4 }
                + P0D &  S2 &  P90D & !S4 & X4{ C5 }
                + P0D &  S2 &  P90D &  S4     { C6 }
                +!P0D &  S2 &  P90D &  S4 & X4{ C7 }
                + P0D & !S2 & !P90D & !S4 & X4{ C8 }
                ) & !RESET;
        UP :=
            (
                !P0D &  S2 & !P90D &  S4
                +!P0D &  S2 &  P90D &  S4
                +!P0D &  S2 &  P90D & !S4
                + P0D &  S2 &  P90D & !S4
                + P0D & !S2 &  P90D & !S4
                + P0D & !S2 & !P90D & !S4
                + P0D & !S2 & !P90D &  S4
                +!P0D & !S2 & !P90D &  S4
                ) & !RESET;
END;

END QuadDivider;
```


© 1997 Microchip Technology Inc.

APPENDIX C:

C.1 PID ALGORITHM FLOWCHART



© 1997 Microchip Technology Inc.

C.2 PID ALGORITHM CODE LISTING

```
;*****************************************************************************
; NAME:         doServo
;
; DESCRIPTION: Performs the servo loop calculations.
;
doServo

        MOV16       POSERROR,U0         ; save new position error in U0

        LOADAB      U0,KP               ; compute KP*U0                          ┐
        CALL        Dmult                                                        │ Basic PID
        MVPF32      DPX,Y               ; Y=KP*U0                                │ calculation
                                                                                 │
        CLRF        WREG                ; if previous output saturated, do       │ anti-
        CPFSGT      SATFLAG             ; not accumulate integrator              │ wind-
        CALL        doIntegral                                                   │ up
                                                                                 │
        LOADAB      INTEGRAL,KI         ; compute KI*INTEGRAL                    │
        CALL        Dmult                                                        │
        ADD32       DPX,Y               ; Y=KP*U0+KI*INTEGRAL                    │
                                                                                 │
        MVFP16      U0,AARG             ; compute KV*(U0-U1)                     │
        SUB16       U1,AARG                                                      │
        MVFP16      KV,BARG                                                      │
        CALL        Dmult                                                        │
        ADD32       DPX,Y               ; Y=KP*U0+KI*INTEGRAL+KV*(U0-U1)         ┘

        CLRF        WREG                                                         ┐
        CPFSGT      SHIFTNUM            ; scale Y by SHIFTNUM                    │ Scale Y
        GOTO        grabok              ; Y = Y * (2**SHIFTNUM)                  │
        MOVFP       SHIFTNUM,TMP                                                 │
grabloop                                                                         │
        RLC32       Y                                                            │
        DECFSZ      TMP                                                          │
        GOTO        grabloop                                                     ┘

grabok
        CLRF        SATFLAG
        BTFSC       Y+B3,MSB            ; saturate to middle 16 bits,
        GOTO        negs                ; keeping top 10 bits for PW1DCH
poss                                    ; and PW1DCL                             ┐
        MOVFP       Y+B2,WREG           ; check if Y >= 2**23                    │
        ANDLW       0x80                                                         │ If positive
        IORWF       Y+B3                                                         │ overflow, saturate
        CLRF        WREG                                                         │ y to maximum
        CPFSGT      Y+B3                                                         │ positive number
        GOTO        zero6bits           ; if not, zero 6 bits                    │
                                                                                 │
        INCF        SATFLAG             ; if so, set Y=0x007FFFFF                │
        CLRF        Y+B3                ; clear for debug purposes               │
        MOVLW       0x7F                                                         │
        MOVPF       WREG,Y+B2                                                    │
        SETF        Y+B1                                                         │
        SETF        Y+B0                                                         │
        GOTO        zero6bits                                                    ┘
negs                                                                             ┐
        MOVFP       Y+B2,WREG           ; check if Y <= -2**23                   │
        IORLW       0x7F                                                         │ If negative
        ANDWF       Y+B3                                                         │ overflow, saturate
        SETF        WREG                                                         │ y to maximum
        CPFSLT      Y+B3                                                         │ negative number
        GOTO        zero6bits           ; if not, zero 6 bits                    │
                                                                                 │
        SETF        SATFLAG             ; if so, set Y = 0xFF800000              ┘
```


© 1997 Microchip Technology Inc.

```
        SETF    Y+B3
        CLRF    Y+B2
        BSF     Y+B2,MSB
        CLRF    Y+B1
        CLRF    Y+B0

zero6bits
        MOV24   Y+B1,YPWM+B0        ; move Y to YPWM and zero 6 bits
doTorque                            ; entry point for torque mode
        MOVLW   0xC0
        ANDWF   YPWM+B0

        BTFSC   YPWM+B1,MSB
        GOTO    tmlimit
tplimit                                                    If external
        BTFSS   EXTSTAT,BIT6                               position
        GOTO    mplimitok                                  limits have
        CLR32   YPWM                                       been reached
        GOTO    mplimitok                                  then zero PWM
tmlimit                                                    output
        BTFSS   EXTSTAT,BIT5
        GOTO    mplimitok
        CLR32   YPWM
mplimitok
        MOVLW   PW1DCH_INIT         ; adjustment from bipolar to unipolar
        MOVPF   WREG,TMP+B1         ; for 50% duty cycle
        MOVLW   PW1DCL_INIT
        MOVPF   WREG,TMP+B0
        ADD16   TMP,YPWM

        CLRF    TMP+B1              ; correct by 1 LSB
        MOVLW   0x40                ; add one to bit5 of PW1DCL
        MOVPF   WREG,TMP+B0
        ADD16   TMP,YPWM

testmax
        CLRF    TMP+B2              ; check pwm maximum limit
        CLRF    YPWM+B2             ; LMD18200 must have a minimum pulse
        CLRF    YPWM+B3             ; so duty cycle must not be 0 or 100%
        MVFP16  YPWMAX,TMP
        SUB24   YPWM,TMP
        BTFSS   TMP+B2,MSB
        GOTO    testmin
        MOV16   YPWMAX,YPWM         ; saturate to max
        GOTO    limitok
testmin
        CLRF    TMP+B2              ; check pwm minimum limit
        CLRF    YPWM+B2
        CLRF    YPWM+B3
        MVFP16  YPWMIN,TMP
        SUB24   YPWM,TMP
        BTFSC   TMP+B2,MSB
        GOTO    limitok
        MOV16   YPWMIN,YPWM         ; saturate to min
limitok
        MOVLB   BANK3               ; set new duty cycle
        MOVFP   YPWM+B0,PW1DCL
        MOVFP   YPWM+B1,PW1DCH

        MOV16   U0,U1               ; push errors into U(k-1)

        RETURN

;*******************************************************************************
```

Convert PWM from unnipolar to bipolar

PWM cycle must not be 0% of 100%

Write PWM values to PWM registers

4

APPENDIX D: ENCODER INTERFACE ROUTINE

```
;*****************************************************************************
; NAME:         doMPosMVel
;
; DESCRIPTION: Calculates current position from UpCount and DownCount
;

doMPosMVel

; Do UpCounter first

        MVFP16   UPCOUNT,TMP+B0             ; save old upcount
readUp
        MOVPF    TMR0H,WREG
        MOVPF    TMR0L,UPCOUNT+B0
        CPFSEQ   TMR0H                      ; Skip next if HI hasn't changed
        GOTO     readUp                     ; HI changed, re-read LO
        MOVPF    WREG,UPCOUNT+B1            ; OK to store HI now

        CLRFM    VELOCITY+B0                ; clear bits below binary point

        MOV16    UPCOUNT,MVELOCITY+B1       ; compute upcount increment
        SUB16    TMP+B0,MVELOCITY+B1

; Now do DownCounter

        MVFP16   DOWNCOUNT,TMP+B0           ; save old downcount
readDown
        MOVLB    BANK2                      ;timers in Bank 2
        MOVPF    TMR3H,WREG
        MOVPF    TMR3L,DOWNCOUNT+B0
        CPFSEQ   TMR3H                      ; Skip next if HI hasn't changed
        GOTO     readDown                   ; HI changed, re-read LO
        MOVPF    WREG,DOWNCOUNT+B1          ; OK to store HI now

        MVFP16   DOWNCOUNT+B0,TMP+B2        ; compute downcount increment
        SUB16    TMP+B0,TMP+B2

        SUB16    TMP+B2,MVELOCITY+B1        ; compute new measured velocity

        CLRF     MVELOCITY+B3               ; sign extend measured velocity for
        BTFSC    MVELOCITY+B2,MSB           ; 24 bit addition to measured position
        SETF     MVELOCITY+B3



        ADD24    MVELOCITY+B1,MPOSITION     ; compute new measured position
                                            ; delta position = measured velocity

        RETURN

;*****************************************************************************
```


© 1997 Microchip Technology Inc.

APPENDIX E: IMPLEMENTATION DETAILS OF TRAJECTORY GENERATION

doPreMove

This routine is executed only once at the beginning of each move. First, various buffers and flags are initialized and a test for modetype is performed. In position mode, the minimum move is triangular and consists of two steps. Therefore, if abs (MOVVAL) > 2, an immediate move is performed. Otherwise, normal move generation is possible with the sign of the move in MOVSIGN and the appropriate signed velocity and acceleration limits in V and A, and MOVVAL/2 in HMOVVAL.

In velocity mode, the sign of the move is calculated in MOVSIGN and the appropriate signed velocity and acceleration limits are placed in V and A. Finally, at modeready, MOVVAL is sign extended for higher precision arithmetic and the servo is enabled.

In torque mode, MOVVAL is output directly to the PWM and the servo is disabled, and doMove is not executed.

doMove

Move generation is based on a piecewise constant acceleration model. During constant acceleration, this results in the standard equations for position and velocity given by:

$$x(t) = x0 + v0 \times t + a \times = (t \times 2)/2, v(t) = v0 + a \times t$$

With the units for t in sample times, the time increment between subsequent sample times is 1, yielding the iterative equations for updating position and velocity implemented in doPosVel and given by:

$$P(k) = P(k-1) + V(k-1) + A/2, V(k) = V(k-1) + A$$

where A is the signed acceleration limit calculated in doPreMove. The inverse equations of this iteration, necessary for undoing an unwanted step, are contained in undoPosVel and given by:

$$P(k-1) = P(k) - V(K-1) - A/2, V(K-1) = V(k) - A$$

In position mode, the actual shape of the velocity profile depends on the values of V, A, and the size of the move. Either the velocity limit is reached before half the move is completed, resulting in a trapezoidal velocity profile, or half the move is completed before the velocity limit is realized, resulting in a triangular velocity profile.

In the algorithm employed here, the velocity limit is treated as a bound on the actual velocity limit, thereby permitting exactly the same number of steps during the speedup and speed down sections of the move. Phase 1 is defined as the section of the move where the commanded position is less than half the move, and phase 2 is the remaining portion of the move. T1 is time when the actual velocity limit is reached and T2 is the time at the end of phase 1.

4

FIGURE 12:

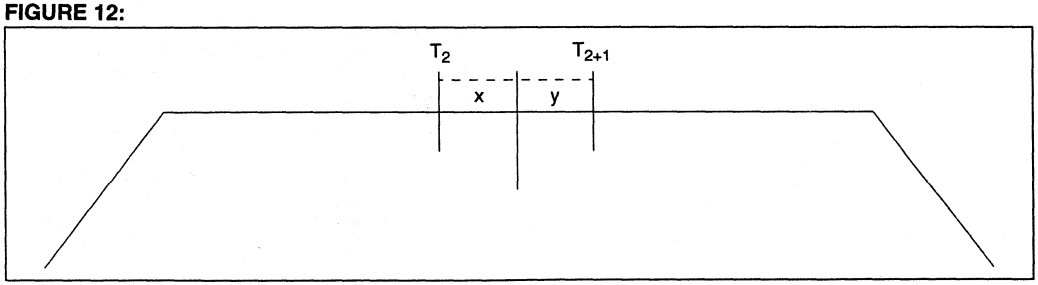


FIGURE 13:

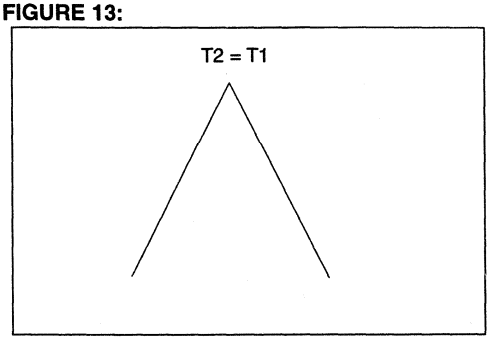
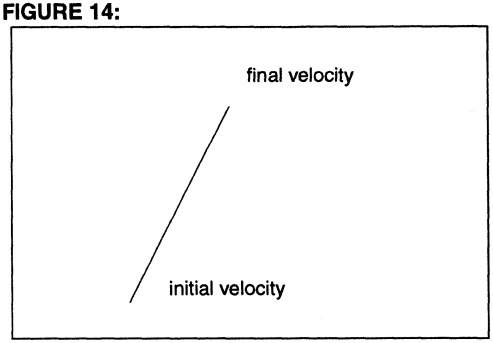


FIGURE 14:



© 1997 Microchip Technology Inc.

Furthermore, let x be the amount of undershoot and y the amount of overshoot of half the move at T2. Discretization error is minimized by using the values of x and y whether one more step will reduce the size of the final immediate move during the last step of the move. For a triangular move, the discretization error is given by min. (2x, 2y), resulting in the condition that if 2x > 2y, then take one more speedup step. In the case of a trapezoidal move, the discretization error is given by min. (2x, y - x), yielding the condition that if 3x > y, take one more step during the flat section of phase2.

At the beginning of doMove, MOVTIME is incremented and doPosVel is called to evaluate the next proposed values of commanded position and velocity under the current value of A. In position mode, phase1, the original position plus half the move minus the new proposed commanded position is calculated and placed in MOVDEL, with the previous MOVDEL saved in MOVTMP. As half the move would be passed, MOVTMP = -x and MOVDEL = y, with y > 0 for the first time indicating that phase1 is about to be completed. Therefore, if y < 0, we continue in phase1, where if maximum velocity has not been reached, the new proposed commanded position is executed. On the other hand, if the proposed move would exceed the maximum velocity, we undo the proposed move, set the current acceleration to zero, reevaluate the iterative equations with the new acceleration, set T1 = MOVTIME - 1, and execute the move.

Since T1 is cleared in doPreMove, it is used as a flag to indicate if this corner in the velocity profile has been reached. Once we find that y > 0, we drop into code that is executed only one time, with phase2 beginning on the next step. If T1 = 0, maximum velocity has not yet been reached, so T1 = T2 and the velocity profile is triangular. In this case, A is negated for speed down, and if x>y, one more step is needed to minimize the discretization error. So A is negated, the proposed step undone, A is again negated for speed down and the step recalculated and executed, with T2 = T1=MOVTIME - 1.

If T1 is not zero, indicating that we are in the flat section of phase1, then go to t2net1, where T2 = MOVTIME - 1, and if 3x > y, then one more phase2 flat step is necessary to minimize the discretization error. PH2FLAT is defined as the number of steps in the flat section of phase2, and is used as a counter during its completion. If 3x > y, then PH2FLAT = T2-T1, otherwise PH2FLAT = T2-T1-1 and phase1 is finally complete. All subsequent steps will proceed through phase2, first deciding if the flat section is finished by checking if PH2FLAT has reached zero. If not, go to flat where PH2FLAT is decremented, and tested if zero. If so, the speed down section is begun by calculating the appropriate signed acceleration limit A, and executing the last of the flat section moves. For all following steps, PH2FLAT = 0, leaving only the final test for zero commanded velocity to indicate the end of the move. This will always occur since the actual maximum velocity, bounded above by the user supplied limit, is always an integer multiple of the user supplied acceleration limit, with exactly the same number of steps taken during speedup and speed down.

The velocity mode is much more straightforward, with the velocity profile in the form of a ramp. If the final velocity has not been reached, the move continues at maximum acceleration. If the final velocity has been reached, the acceleration is set to zero and the move generation of commanded position and velocity continued unless the final velocity is zero.

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX F: COMPLETE CODE LISTING (DCMOTOR.LST)

```
MPASM 01.40 Released          DCMOTOR.ASM   1-16-1997  13:20:16         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         TITLE "DCMOTOR SERVO CONTROL:  Revision:   1.9
                     00002 ;       Revised: 8/5/92
                     00003 ;
                     00004 ;       Program:          DCMOTOR.ASM
                     00005 ;       Revision Date:
                     00006 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00007 ;
                     00008 ;       CREDIT:  Developed by Teknic Inc. 1992
                     00009 ;
                     00010 ;*******************************************************************************
                     00011
                     00012 ;       PROCESSOR       PIC17C42
                     00013         LIST            P = 17C42, COLUMNS=120, XREF=YES, NOWRAP, LINES=255, R=DEC
                     00014
                     00015         #include "dcmotor.h17"
                     00001 ;*******************************************************************************
                     00002 ;
                     00003 ; Header file for dcmotor.asm:
                     00004 ; Revised: 8/5/92
                     00005 ;*******************************************************************************
                     00006 ;
                     00007 ; hardware constants
                     00008 ;
                     00009 ;
00F42400             00010 MASTER_CLOCK    set    16000000        ; 16 MHz: change for diff clock speed
003D0900             00011 CLKOUT          set    MASTER_CLOCK/4
000003E8             00012 SAMPLE_RATE     set    1000
                     00013
0000000C             00014 BAUD19200       set    (MASTER_CLOCK/((32*19200)-1)/2-1)
00000019             00015 BAUD9600        set    (MASTER_CLOCK/((32*9600)-1)/2-1)
00000067             00016 BAUD2400        set    (MASTER_CLOCK/((32*2400)-1)/2-1)
000000CF             00017 BAUD1200        set    (MASTER_CLOCK/((32*1200)-1)/2-1)
000000FF             00018 BAUD_MIN        set    0xFF
00000019             00019 BAUD_DEFAULT    set    BAUD9600
                     00020
```

```
00000006          00021 TCON1_INIT      set     0x06
0000003F          00022 TCON2_INIT      set     0x3F
000000FF          00023 PR1_INIT        set     0xFF            ; set pwm frequency to CLKOUT/256 khz
0000000F          00024 PR2_INIT        set     (CLKOUT/(PR1_INIT+1)+SAMPLE_RATE/2)/SAMPLE_RATE-1
0000007F          00025 PW1DCH_INIT     set     (PR1_INIT/2)    ; set duty cycle to 50%, PW1DCH = PR1_INIT/2
000000C0          00026 PW1DCL_INIT     set     0xC0            ; and PW1DCL = 0xC0
00000080          00027 RTCSTA_INIT     set     0x80
00000090          00028 RCSTA_INIT      set     0x90
00000020          00029 TXSTA_INIT      set     0x20
00000019          00030 SPBRG_INIT      set     BAUD_DEFAULT
                  00031
000000F3          00032 DDRB_INIT       set     0xF3
00000000          00033 DDRD_INIT       set     0x00
                  00034 ;
                  00035 ;
                  00036 ;       max and min pwm values
                  00037 ;
00000040          00038 PWMINL          set     0x40
00000001          00039 PWMINH          set     0x01            ; 0x0000 + 0x0140 (min 10 bit pwm +5)
00000080          00040 PWMAXL          set     0x80
000000FE          00041 PWMAXH          set     0xFE            ; 0xFFC0 - 0x0140 ( max 10 bit pwm -5)
                  00042 ;
                  00043 ;
                  00044 ;
                  00045 ;     17c42 constants
                  00046 ;
                  00047 ;
                  00048 ;
00000000          00049 LO      EQU     0
00000001          00050 HI      EQU     1
00000000          00051 B0      EQU     0
00000001          00052 B1      EQU     1
00000002          00053 B2      EQU     2
00000003          00054 B3      EQU     3
00000007          00055 MSB     EQU     7
00000000          00056 LSB     EQU     0
                  00057 ;
                  00058 ; define special function registers:
                  00059
                  00060         #define W 0
                  00061         #define true    1
                  00062         #define false   0
                  00063         #define TRUE    1
                  00064         #define FALSE   0
                  00065
                  00066         cblock 0x00
00000000          00067                 BIT0,BIT1,BIT2,BIT3,BIT4,BIT5,BIT6,BIT7
```


© 1997 Microchip Technology Inc.

```
                00068         endc
                00069
                00070         cblock 0x00     ; define banks
00000000        00071                BANK0,BANK1,BANK2,BANK3
                00072         endc
                00073
                00074         cblock 0x00             ; unbanked registers
00000000        00075                INDF0,FSR0,PCL,PCLATH,ALUSTA,RTCSTA,CPUSTA,INTSTA
00000008        00076           INDF1,FSR1,WREG,TMR0L,TMR0H,TBLPTRL,TBLPTRH,BSR
                00077         endc
                00078
                00079         cblock 0x10             ; bank0 registers
00000010        00080                PORTA,DDRB,PORTB,RCSTA,RCREG,TXSTA,TXREG,SPBRG
                00081         endc
                00082
                00083         cblock 0x10             ; bank1 registers
00000010        00084                          DDRC,PORTC,DDRD,PORTD,DDRE,PORTE,PIR,PIE
                00085         endc
                00086
                00087         cblock 0x10             ; bank2 registers
00000010        00088                       TMR1,TMR2,TMR3L,TMR3H,PR1,PR2,PR3L,PR3H
                00089         endc
                00090
00000016        00091 CA1L    equ     0x16            ; alternate function def
00000017        00092 CA1H    equ     0x17
                00093
                00094         cblock  0x10     ; define bank3 variables
00000010        00095                       PW1DCL,PW2DCL,PW1DCH,PW2DCH,CA2L,CA2H,TCON1,TCON2
                00096         endc
                00097
                00098 ;*****************************************************************
                00099 ; define commonly used bits:
                00100
                00101 ; ALUSTA bit definitions
                00102
                00103         #define _carry  ALUSTA,0
                00104         #define _c      ALUSTA,0
                00105         #define _cy     ALUSTA,0
                00106         #define _dc     ALUSTA,1
                00107         #define _z      ALUSTA,2
                00108         #define _ov     ALUSTA,3
                00109         #define _fs0    ALUSTA,4
                00110         #define _fs1    ALUSTA,5
                00111         #define _fs2    ALUSTA,6
                00112         #define _fs3    ALUSTA,7
                00113
                00114 ; T0STA bit definitions
```

© 1997 Microchip Technology Inc.

```
00115
00116          #define _ps0    T0STA,1
00117          #define _ps1    T0STA,2
00118          #define _ps2    T0STA,3
00119          #define _ps3    T0STA,4
00120          #define _tosc   T0STA,5
00121          #define _tose   T0STA,6
00122          #define _intedg T0STA,7
00123
00124 ; CPUSTA bit definitions
00125
00126          #define _npd    CPUSTA,2
00127          #define _nto    CPUSTA,3
00128          #define _gint   CPUSTA,4
00129          #define _glintd CPUSTA,4
00130          #define _stkav  CPUSTA,5
00131
00132 ; INTSTA bit definitions
00133
00134          #define _inte   INTSTA,0
00135          #define _toie   INTSTA,1
00136          #define _t0ckie INTSTA,2
00137          #define _peie   INTSTA,3
00138          #define _intf   INTSTA,4
00139          #define _t0if   INTSTA,5
00140          #define _t0ckif INTSTA,6
00141          #define _peif   INTSTA,7
00142
00143 ; PIR Bit definitions
00144
00145          #define _rcif   PIR,0
00146          #define _txif   PIR,1
00147          #define _ca1if  PIR,2
00148          #define _ca2if  PIR,3
00149          #define _tmr1if PIR,4
00150          #define _tmr2if PIR,5
00151          #define _tmr3if PIR,6
00152          #define _rbif   PIR,7
00153
00154
00155 ; PIE Bit definitions
00156
00157          #define _rcie   PIE,0
00158          #define _txie   PIE,1
00159          #define _ca1ie  PIE,2
00160          #define _ca2ie  PIE,3
00161          #define _tmr1ie  PIE,4
```


© 1997 Microchip Technology Inc.

```
00162         #define _tmr2ie PIE,5
00163         #define _tmr3ie PIE,6
00164         #define _rbie    PIE,7
00165
00166 ; RCSTA bit definitions
00167
00168         #define _rx9d   RCVSTA,0
00169         #define _oerr   RCVSTA,1
00170         #define _ferr   RCVSTA,2
00171         #define _cren   RCVSTA,4
00172         #define _cren   RCVSTA,5
00173         #define _rx9    RCVSTA,6
00174         #define _spen   RCVSTA,7
00175
00176 ; TXSTA bit definitions
00177
00178         #define _tx9d   TXSTA,0
00179         #define _trmt   TXSTA,1
00180         #define _sync   TXSTA,4
00181         #define _txen   TXSTA,5
00182         #define _tx9    TXSTA,6
00183         #define _csrc   TXSTA,7
00184
00185 ; TCON1 bit definitions
00186
00187         #define _tmr1cs TCON1,0
00188         #define _tmr2cs TCON1,1
00189         #define _tmr3cs TCON1,2
00190         #define _t16    TCON1,3
00191         #define _ca1ed0 TCON1,4
00192         #define _ca1ed1 TCON1,5
00193         #define _ca2ed0 TCON1,6
00194         #define _ca2ed1 TCON1,7
00195
00196 ; TCON2 bit definitions
00197
00198         #define _tmr1on  TCON2,0
00199         #define _tmr2on  TCON2,1
00200         #define _tmr3on  TCON2,2
00201         #define _ca1pr3  TCON2,3
00202         #define _pwm1on  TCON2,4
00203
00204         #define _pwm2on  TCON2,5
00205         #define _ca1ovf  TCON2,6
00206         #define _ca2ovf  TCON2,7
00207 ;
00208 ;
```


© 1997 Microchip Technology Inc.

```
                    00209 ;
                    00210 ;     ascii constants
                    00211 ;
                    00212 ;
0000000D            00213 CR      set    0x0D
00000018            00214 CAN     set    0x18
00000008            00215 BS      set    0x08
00000020            00216 SP      set    0x20
0000000A            00217 LF      set    0x0A
0000002D            00218 MN      set    '-'
                    00219 ;
                    00220 ;
                    00221 ;*******************************************************************************
                    00222 ;
00000001            00223 DECIO   EQU    TRUE                    ; true for decimal, false for hex
                    00224 ;
                    00225 ;     cmds constants and macros
                    00226 ;
                    00227 ;
00000001            00228 CHARREADY       set    0x01
                    00229 ;
                    00230 ;
00000008            00231 NUMPAR          set    0x08
                    00232 ;
                    00233 ; Response characters
                    00234 ;
00000021            00235 CMD_OK          set    '!'
0000003F            00236 CMD_BAD         set    '?'
                    00237 ;
                    00238 ; Exit values
                    00239 ;
                    00240 ;
00000000            00241 HEX_SP          set    0x00
00000001            00242 HEX_MN          set    0x01
00000002            00243 HEX_CR          set    0x02
00000003            00244 HEX_CAN         set    0x03
                    00245 ;
00000000            00246 DEC_SP          set    0x00
00000001            00247 DEC_MN          set    0x01
00000002            00248 DEC_CR          set    0x02
00000003            00249 DEC_CAN         set    0x03
                    00250 ;
                    00251 ;
                    00252 ; Command characters
                    00253 ;
0000000D            00254 DO_NULL                set    CR
0000004D            00255 DO_MOVE                set    'M'     ; M
```


© 1997 Microchip Technology Inc.

```
0000004F        00256 DO_MODE                 set     'O'     ; O
00000053        00257 DO_SETPARAMETER         set     'S'     ; S
00000052        00258 DO_READPARAMETER        set     'R'     ; R
00000043        00259 DO_SHUTTER              set     'C'     ; C
00000050        00260 DO_READCOMPOSITION      set     'P'     ; P
00000056        00261 DO_READCOMVELOCITY      set     'V'     ; V
00000070        00262 DO_READACTPOSITION      set     'p'     ; p
00000076        00263 DO_READACTVELOCITY      set     'v'     ; v
00000058        00264 DO_EXTERNALSTATUS       set     'X'     ; X
00000059        00265 DO_MOVESTATUS           set     'Y'     ; Y
00000049        00266 DO_READINDPOSITION      set     'I'     ; I
00000048        00267 DO_SETPOSITION          set     'H'     ; H
0000005A        00268 DO_RESET                set     'Z'     ; Z
00000073        00269 DO_STOP                 set     's'     ; s
00000063        00270 DO_CAPTURE              set     'c'     ; c
                00271
                00272 ;*****************************************************************************
                00273 ; NAME:          CMD_DEF
                00274 ;
                00275 ; DESCRIPTION:   Creates all the definitions for a command table data struc-
                00276 ;               ture. The first word is at the command character used, and
                00277 ;               the second word is a pointer to the function that handles
                00278 ;               this command function.
                00279 ;
                00280 ; ENTRY CONDITIONS:     Must be contiguous with the other entries for the
                00281 ;                       function to work.
                00282 ;
                00283 ; ARGUMENTS:            FUNC    command execution function
                00284 ;                       ROOT    NAME ROOT
                00285 ;
                00286
                00287 CMD_DEF MACRO   FUNC,ROOT
                00288
                00289         DATA    ROOT
                00290         DATA    FUNC
                00291         ENDM
                00292
00000002        00293 CMD_ENTRY_LENGTH        set     2
                00294
                00295 ;*****************************************************************************
                00296
                00297 ;*****************************************************************************
                00298 ; NAME:          CMD_START
                00299 ;
                00300 ; DESCRIPTION:   Labels the start of the command table.
                00301 ;
                00302
```

© 1997 Microchip Technology Inc.

```
00303 CMD_START MACRO LABEL
00304
00305 LABEL
00306         ENDM                                    ;
00307
00308 ;*****************************************************************************
00309
00310 ;*****************************************************************************
00311 ; NAME:          CMD_END
00312 ;
00313 ; DESCRIPTION:   Marks the end of the command table with an entry of 0x00
00314 ;
00315
00316 CMD_END MACRO
00317 ;                                               ;
00318         DATA    0x00
00319         ENDM                                    ;
00320
00321 ;*****************************************************************************
00322
00323 ;*****************************************************************************
00324 ; NAME:          CLR32
00325 ;
00326 ; DESCRIPTION:   Clear 4 consecutive bytes of data memory
00327 ;
00328 ; ARGUMENTS:     0 => a
00329 ;
00330 ; TIMING (cycles): 4
00331 ;
00332
00333 CLR32 MACRO     a
00334         CLRF    a+B0, F
00335         CLRF    a+B1, F
00336         CLRF    a+B2, F
00337         CLRF    a+B3, F
00338
00339         ENDM                                    ;
00340
00341 ;*****************************************************************************
00342
00343 ;*****************************************************************************
00344 ; NAME:          CLR24
00345 ;
00346 ; DESCRIPTION:   Clear 3 consecutive bytes of data memory
00347 ;
00348 ; ARGUMENTS:     0 => a
00349 ;
```


© 1997 Microchip Technology Inc.

```
00350 ; TIMING (cycles): 3
00351 ;
00352
00353 CLR24 MACRO     a
00354         CLRF    a+B0, F
00355         CLRF    a+B1, F
00356         CLRF    a+B2, F
00357
00358         ENDM
00359
00360 ;*****************************************************************************
00361
00362 ;*****************************************************************************
00363 ; NAME:          CLR16
00364 ;
00365 ; DESCRIPTION:   Clear 2 consecutive bytes of data memory
00366 ;
00367 ; ARGUMENTS:     0 => a
00368 ;
00369 ; TIMING(cycles): 2
00370 ;
00371
00372 CLR16 MACRO     a
00373         CLRF    a+B0, F
00374         CLRF    a+B1, F
00375
00376         ENDM
00377
00378 ;*****************************************************************************
00379
00380 ;*****************************************************************************
00381 ; NAME:          MOV32
00382 ;
00383 ; DESCRIPTION:   32 bit move
00384 ;
00385 ; ARGUMENTS:     a => b
00386 ;
00387 ; TIMING (cycles):8
00388 ;
00389
00390 MOV32 MACRO     a,b
00391
00392         MOVFP   a+B0,WREG               ; get byte of a into w
00393         MOVPF   WREG,b+B0               ; move to b(B0)
00394         MOVFP   a+B1,WREG               ; get byte of a into w
00395         MOVPF   WREG,b+B1               ; move to b(B1)
00396         MOVFP   a+B2,WREG               ; get byte of a into w
```


© 1997 Microchip Technology Inc.

```
00397         MOVPF   WREG,b+B2               ; move to b(B2)
00398         MOVFP   a+B3,WREG               ; get byte of a into w
00399         MOVPF   WREG,b+B3               ; move to b(B3)
00400
00401         ENDM
00402
00403 ;*******************************************************************************
00404
00405 ;*******************************************************************************
00406 ; NAME:          MOV24
00407 ;
00408 ; DESCRIPTION:  24 bit move
00409 ;
00410 ; ARGUMENTS:    a => b
00411 ;
00412 ; TIMING (cycles): 6
00413 ;
00414
00415 MOV24 MACRO     a,b
00416
00417         MOVFP   a+B0,WREG               ; get byte of a into w
00418         MOVPF   WREG,b+B0               ; move to b(B0)
00419         MOVFP   a+B1,WREG               ; get byte of a into w
00420         MOVPF   WREG,b+B1               ; move to b(B1)
00421         MOVFP   a+B2,WREG               ; get byte of a into w
00422         MOVPF   WREG,b+B2               ; move to b(B2)
00423
00424         ENDM
00425
00426 ;*******************************************************************************
00427
00428 ;*******************************************************************************
00429 ; NAME:          MOV16
00430 ;
00431 ; DESCRIPTION:  16 bit move
00432 ;
00433 ; ARGUMENTS:    a => b
00434 ;
00435 ; TIMING (in cycles): 4
00436 ;
00437
00438 MOV16 MACRO     a,b
00439
00440         MOVFP   a+B0,WREG               ; get byte of a into w
00441         MOVPF   WREG,b+B0               ; move to b(B0)
00442         MOVFP   a+B1,WREG               ; get byte of a into w
00443         MOVPF   WREG,b+B1               ; move to b(B1)
```


© 1997 Microchip Technology Inc.

```
00444
00445         ENDM
00446
00447 ;*****************************************************************************
00448
00449 ;*****************************************************************************
00450 ; NAME:          MVPF32
00451 ;
00452 ; DESCRIPTION:  32 bit move from P data memory to F data memory
00453 ;
00454 ; ARGUMENTS:     A => B
00455 ;
00456 ; TIMING (cycles): 4
00457 ;
00458
00459 MVPF32 MACRO     A,B
00460
00461         MOVPF    A+B0,B+B0              ; move A(B0) to B(B0)
00462         MOVPF    A+B1,B+B1              ; move A(B1) to B(B1)
00463         MOVPF    A+B2,B+B2              ; move A(B2) to B(B2)
00464         MOVPF    A+B3,B+B3              ; move A(B3) to B(B3)
00465
00466         ENDM
00467
00468 ;*****************************************************************************
00469
00470 ;*****************************************************************************
00471 ; NAME:          MVPF24
00472 ;
00473 ; DESCRIPTION:  24 bit move from P data memory to F data memory
00474 ;
00475 ; ARGUMENTS:     A => B
00476 ;
00477 ;
00478 ; TIMING (cycles): 3
00479 ;
00480
00481 MVPF24 MACRO     A,B
00482
00483         MOVPF    A+B0,B+B0              ; move A(B0) to B(B0)
00484         MOVPF    A+B1,B+B1              ; move A(B1) to B(B1)
00485         MOVPF    A+B2,B+B2              ; move A(B2) to B(B2)
00486
00487         ENDM
00488
00489 ;*****************************************************************************
00490
```


© 1997 Microchip Technology Inc.

```
00491 ;*******************************************************************************
00492 ; NAME:          MVPF16
00493 ;
00494 ; DESCRIPTION:  16 bit move from P data memory to F data memory
00495 ;
00496 ; ARGUMENTS:    A => B
00497 ;
00498 ; TIMING (cycles): 2
00499 ;
00500
00501 MVPF16 MACRO    A,B
00502
00503         MOVPF   A+B0,B+B0               ; move A(B0) to B(B0)
00504         MOVPF   A+B1,B+B1               ; move A(B1) to B(B1)
00505
00506         ENDM
00507
00508 ;*******************************************************************************
00509
00510
00511 ;*******************************************************************************
00512 ; NAME:          MVFP32
00513 ;
00514 ; DESCRIPTION:  32 bit move from F data memory to P data memory
00515 ;
00516 ; ARGUMENTS:    A => B
00517 ;
00518 ; TIMING (cycles): 4
00519
00520 MVFP32 MACRO    A,B
00521
00522         MOVFP   A+B0,B+B0               ; move A(B0) to B(B0)
00523         MOVFP   A+B1,B+B1               ; move A(B1) to B(B1)
00524         MOVFP   A+B2,B+B2               ; move A(B2) to B(B2)
00525         MOVFP   A+B3,B+B3               ; move A(B3) to B(B3)
00526
00527         ENDM
00528
00529 ;*******************************************************************************
00530
00531 ;*******************************************************************************
00532 ; NAME:          MVFP24
00533 ;
00534 ; DESCRIPTION:  24 bit move from F data memory to P data memory
00535 ;
00536 ; ARGUMENTS:    A => B
00537 ;
```


© 1997 Microchip Technology Inc.

```
00538 ; TIMING (cycles): 3
00539 ;
00540
00541 MVFP24 MACRO    A,B
00542
00543         MOVFP   A+B0,B+B0               ; move A(B0) to B(B0)
00544         MOVFP   A+B1,B+B1               ; move A(B1) to B(B1)
00545         MOVFP   A+B2,B+B2               ; move A(B2) to B(B2)
00546
00547         ENDM
00548
00549 ;*******************************************************************************
00550
00551 ;*******************************************************************************
00552 ; NAME:          MVFP16
00553 ;
00554 ; DESCRIPTION:  16 bit move from F data memory to P data memory
00555 ;
00556 ; ARGUMENTS:    A => B
00557 ;
00558 ; TIMING (cycles): 2
00559 ;
00560
00561 MVFP16 MACRO    A,B
00562
00563         MOVFP   A+B0,B+B0               ; move A(B0) to B(B0)
00564         MOVFP   A+B1,B+B1               ; move A(B1) to B(B1)
00565
00566         ENDM
00567
00568 ;*******************************************************************************
00569
00570 ;*******************************************************************************
00571 ; NAME:          LOADAB
00572 ;
00573 ; DESCRIPTION:  Loads extended math library AARG and BARG
00574 ;
00575 ; ARGUMENTS:    A => AARG
00576 ;               B => BARG
00577 ;
00578 ; TIMING (cycles): 4
00579
00580 LOADAB MACRO    A,B
00581
00582         MOVFP   A+B0,AARG+B0            ; load lo byte of A to AARG
00583         MOVFP   A+B1,AARG+B1            ; load hi byte of A to AARG
00584         MOVFP   B+B0,BARG+B0            ; load lo byte of B to BARG
```


© 1997 Microchip Technology Inc.

```
00585         MOVFP   B+B1,BARG+B1            ; load hi byte of B to BARG
00586
00587         ENDM
00588
00589 ;*****************************************************************************
00590
00591 ;*****************************************************************************
00592 ; NAME:          ADD32
00593 ;
00594 ; DESCRIPTION:   32 bit add
00595 ;
00596 ; ARGUMENTS:     a + b => b
00597 ;
00598 ; TIMING (cycles): 8
00599 ;
00600
00601 ADD32 MACRO     a,b
00602
00603         MOVFP   a+B0,WREG               ; get lowest byte of a into w
00604         ADDWF   b+B0, F                 ; add lowest byte of b, save in b(B0)
00605         MOVFP   a+B1,WREG               ; get 2nd byte of a into w
00606         ADDWFC  b+B1, F                 ; add 2nd byte of b, save in b(B1)
00607         MOVFP   a+B2,WREG               ; get 3rd byte of a into w
00608         ADDWFC  b+B2, F                 ; add 3rd byte of b, save in b(B2)
00609         MOVFP   a+B3,WREG               ; get 4th byte of a into w
00610         ADDWFC  b+B3, F                 ; add 4th byte of b, save in b(B3)
00611
00612         ENDM
00613
00614 ;*****************************************************************************
00615
00616 ;*****************************************************************************
00617 ; NAME:          ADD24
00618 ;
00619 ; DESCRIPTION:   24 bit add
00620 ;
00621 ; ARGUMENTS:     a + b => b
00622 ;
00623 ; TIMING (cycles): 6
00624 ;
00625
00626 ADD24 MACRO     a,b
00627
00628         MOVFP   a+B0,WREG               ; get lowest byte of a into w
00629         ADDWF   b+B0, F                 ; add lowest byte of b, save in b(B0)
00630         MOVFP   a+B1,WREG               ; get 2nd byte of a into w
00631         ADDWFC  b+B1, F                 ; add 2nd byte of b, save in b(B1)
```


© 1997 Microchip Technology Inc.

```
00632         MOVFP   a+B2,WREG               ; get 3rd byte of a into w
00633         ADDWFC  b+B2, F                 ; add 3rd byte of b, save in b(B2)
00634
00635         ENDM
00636
00637 ;*****************************************************************************
00638
00639 ;*****************************************************************************
00640 ; NAME:          ADD16
00641 ;
00642 ; DESCRIPTION:   16 bit add
00643 ;
00644 ; ARGUMENTS:     a + b => b
00645 ;
00646 ;
00647 ; TIMING (cycles): 4
00648 ;
00649
00650 ADD16 MACRO     a,b
00651
00652         MOVFP   a+B0,WREG               ; get lowest byte of a into w
00653         ADDWF   b+B0, F                 ; add lowest byte of b, save in b(B0)
00654         MOVFP   a+B1,WREG               ; get 2nd byte of a into w
00655         ADDWFC  b+B1, F                 ; add 2nd byte of b, save in b(B1)
00656
00657         ENDM
00658
00659 ;*****************************************************************************
00660
00661 ;*****************************************************************************
00662 ; NAME:          SUB32
00663 ;
00664 ; DESCRIPTION:   32 bit subtract
00665 ;
00666 ;
00667 ; ARGUMENTS:     b - a => b
00668 ;
00669 ; TIMING (cycles): 8
00670 ;
00671
00672 SUB32 MACRO     a,b
00673
00674         MOVFP   a+B0,WREG               ; get lowest byte of a into w
00675         SUBWF   b+B0, F                 ; sub lowest byte of b, save in b(B0)
00676         MOVFP   a+B1,WREG               ; get 2nd byte of a into w
00677         SUBWFB  b+B1, F                 ; sub 2nd byte of b, save in b(B1)
00678         MOVFP   a+B2,WREG               ; get 3rd byte of a into w
```


© 1997 Microchip Technology Inc.

```
00679         SUBWFB  b+B2, F                 ; sub 3rd byte of b, save in b(B2)
00680         MOVFP   a+B3,WREG               ; get 4th byte of a into w
00681         SUBWFB  b+B3, F                 ; sub 4th byte of b, save in b(B3)
00682
00683         ENDM
00684
00685 ;*****************************************************************************
00686
00687 ;*****************************************************************************
00688 ; NAME:          SUB24
00689 ;
00690 ; DESCRIPTION:  24 bit subtract
00691 ;
00692 ; ARGUMENTS:    b - a => b
00693 ;
00694 ; TIMING (in cycles): 6
00695 ;
00696
00697 SUB24 MACRO     a,b
00698
00699         MOVFP   a+B0,WREG               ; get lowest byte of a into w
00700         SUBWF   b+B0, F                 ; sub lowest byte of b, save in b(B0)
00701         MOVFP   a+B1,WREG               ; get 2nd byte of a into w
00702         SUBWFB  b+B1, F                 ; sub 2nd byte of b, save in b(B1)
00703         MOVFP   a+B2,WREG               ; get 3rd byte of a into w
00704         SUBWFB  b+B2, F                 ; sub 3rd byte of b, save in b(B2)
00705
00706         ENDM
00707
00708 ;*****************************************************************************
00709
00710 ;*****************************************************************************
00711 ; NAME:          SUB16
00712 ;
00713 ; DESCRIPTION:  16 bit subtract
00714 ;
00715 ; ARGUMENTS:    b - a => b
00716 ;
00717 ; TIMING (cycles): 4
00718 ;
00719
00720 SUB16 MACRO     a,b
00721
00722         MOVFP   a+B0,WREG               ; get lowest byte of a into w
00723         SUBWF   b+B0, F                 ; sub lowest byte of b, save in b(B0)
00724         MOVFP   a+B1,WREG               ; get 2nd byte of a into w
00725         SUBWFB  b+B1, F                 ; sub 2nd byte of b, save in b(B1)
```


© 1997 Microchip Technology Inc.

```
00726
00727         ENDM
00728
00729 ;*****************************************************************************
00730
00731 ;*****************************************************************************
00732 ; NAME: RLC32
00733 ;
00734 ; DESCRIPTION: 32 bit rotate left
00735 ;
00736 ; ARGUMENTS: 2*a => a
00737 ;
00738 ; TIMING (cycles): 5
00739 ;
00740
00741 RLC32 MACRO     a
00742
00743         BCF     _carry
00744         RLCF    a+B0, F
00745         RLCF    a+B1, F
00746         RLCF    a+B2, F
00747         RLCF    a+B3, F
00748
00749         ENDM
00750
00751 ;*****************************************************************************
00752
00753 ;*****************************************************************************
00754 ; NAME:         RLC24
00755 ;
00756 ; DESCRIPTION:  24 bit rotate left
00757 ;
00758 ; ARGUMENTS:    2*a => a
00759 ;
00760 ; TIMING (cycles): 4
00761 ;
00762
00763 RLC24 MACRO     a
00764
00765         BCF     _carry
00766         RLCF    a+B0, F
00767         RLCF    a+B1, F
00768         RLCF    a+B2, F
00769
00770         ENDM
00771
00772 ;*****************************************************************************
```


© 1997 Microchip Technology Inc.

```
00773
00774 ;*****************************************************************************
00775 ; NAME:          RLC16
00776 ;
00777 ; DESCRIPTION:   16 bit rotate left
00778 ;
00779 ; ARGUMENTS:     2*a => a
00780 ;
00781 ;
00782 ; TIMING (cycles): 3
00783 ;
00784
00785 RLC16 MACRO      a
00786         BCF      _carry
00787         RLCF     a+B0, F
00788         RLCF     a+B1, F
00789
00790         ENDM
00791
00792 ;*****************************************************************************
00793
00794 ;*****************************************************************************
00795 ; NAME:          RRC32
00796 ;
00797 ; DESCRIPTION:   32 bit rotate right
00798 ;
00799 ; ARGUMENTS:     a/2 => a
00800 ;
00801 ; TIMING (cycles): 5
00802 ;
00803
00804 RRC32 MACRO      a
00805
00806         RLCF     a+B3,W                  ; move sign into carry bit
00807         RRCF     a+B3, F
00808         RRCF     a+B2, F
00809         RRCF     a+B1, F
00810         RRCF     a+B0, F
00811
00812         ENDM
00813
00814 ;*****************************************************************************
00815
00816 ;*****************************************************************************
00817 ; NAME:          RRC24
00818 ;
00819 ; DESCRIPTION:   24 bit rotate right
```


© 1997 Microchip Technology Inc.

```
00820 ;
00821 ; ARGUMENTS:    a/2 => a
00822 ;
00823 ; TIMING (cycles): 4
00824 ;
00825
00826 RRC24 MACRO     a
00827
00828         RLCF    a+B2,W                  ; move sign into carry bit
00829         RRCF    a+B2, F
00830         RRCF    a+B1, F
00831         RRCF    a+B0, F
00832
00833         ENDM
00834
00835 ;*****************************************************************************
00836
00837 ;*****************************************************************************
00838 ; NAME:          RRC16
00839 ;
00840 ; DESCRIPTION:  16 bit rotate right
00841 ;
00842 ; ENTRY CONDITIONS: a/2 => a
00843 ;
00844 ; TIMING (cycles): 3
00845 ;
00846
00847 RRC16 MACRO     a
00848
00849         RLCF    a+B1,W                  ; move sign into carry bit
00850         RRCF    a+B1, F
00851         RRCF    a+B0, F
00852
00853         ENDM
00854
00855 ;*****************************************************************************
00856
00857 ;*****************************************************************************
00858 ; NAME:          INC24
00859 ;
00860 ; DESCRIPTION:  24 bit increment
00861 ;
00862 ; ARGUMENTS:    a+1 => a
00863 ;
00864 ; TIMING (cycles): 4
00865 ;
00866
```


© 1997 Microchip Technology Inc.

```
00867 INC24 MACRO     a
00868
00869         CLRF    WREG, F
00870         INCF    a+B0, F
00871         ADDWFC  a+B1, F
00872         ADDWFC  a+B2, F
00873
00874         ENDM
00875
00876 ;*****************************************************************************
00877
00878 ;*****************************************************************************
00879 ; NAME:          INC16
00880 ;
00881 ; DESCRIPTION:  16 bit increment
00882 ;
00883 ; ARGUMENTS:    a+1 => a
00884 ;
00885 ; TIMING (cycles): 3
00886 ;
00887
00888 INC16 MACRO     a
00889
00890         CLRF    WREG, F
00891         INCF    a+B0, F
00892         ADDWFC  a+B1, F
00893
00894         ENDM
00895
00896 ;*****************************************************************************
00897
00898 ;*****************************************************************************
00899 ; NAME:          DEC24
00900 ;
00901 ; DESCRIPTION:  Decrement A 24 Bit Number
00902 ;
00903 ; ARGUMENTS:    a-1 => a
00904 ;
00905 ; TIMING (cycles): 4
00906 ;
00907
00908 DEC24 MACRO     a
00909
00910         CLRF    WREG, F
00911         DECF    a+B0, F
00912         SUBWFB  a+B1, F
00913         SUBWFB  a+B2, F
```


© 1997 Microchip Technology Inc.

```
00914
00915         ENDM
00916
00917 ;*****************************************************************************
00918
00919 ;*****************************************************************************
00920 ; DESCRIPTION:  Decrement A 16 Bit Number
00921 ;
00922 ; ARGUMENTS:    a-1 => a
00923 ;
00924 ; TIMING (cycles): 3
00925 ;
00926
00927 DEC16 MACRO     a
00928
00929         CLRF    WREG, F
00930         DECF    a+B0, F
00931         SUBWFB  a+B1, F
00932
00933         ENDM
00934
00935 ;*****************************************************************************
00936
00937 ;*****************************************************************************
00938 ; NAME:         NEG32
00939 ;
00940 ; DESCRIPTION:  32 bit negate
00941 ;
00942 ; ARGUMENTS:    -A => A
00943 ;
00944 ; TIMING (cycles): 9
00945 ;
00946
00947 NEG32 MACRO     A
00948
00949         COMF    A+B0, F
00950         COMF    A+B1, F
00951         COMF    A+B2, F
00952         COMF    A+B3, F
00953         CLRF    WREG, F
00954         INCF    A+B0, F
00955         ADDWFC  A+B1, F
00956         ADDWFC  A+B2, F
00957         ADDWFC  A+B3, F
00958
00959         ENDM
00960
```

© 1997 Microchip Technology Inc.

```
00961 ;*******************************************************************************
00962
00963 ;*******************************************************************************
00964 ; NAME:          NEG24
00965 ;
00966 ; DESCRIPTION:  24 bit negate
00967 ;
00968 ; ARGUMENTS:    -A => A
00969 ;
00970 ; TIMING (cycles): 7
00971 ;
00972
00973 NEG24 MACRO     A
00974
00975         COMF    A+B0, F
00976         COMF    A+B1, F
00977         COMF    A+B2, F
00978         CLRF    WREG, F
00979         INCF    A+B0, F
00980         ADDWFC  A+B1, F
00981         ADDWFC  A+B2, F
00982
00983         ENDM
00984
00985 ;*******************************************************************************
00986
00987 ;*******************************************************************************
00988 ; NAME:          NEG16
00989 ;
00990 ; DESCRIPTION:  16 bit negate
00991 ;
00992 ; ARGUMENTS:    -A => A
00993 ;
00994 ; TIMING (cycles): 5
00995 ;
00996
00997 NEG16 MACRO     A
00998
00999         COMF    A+B0, F
01000         COMF    A+B1, F
01001         CLRF    WREG, F
01002         INCF    A+B0, F
01003         ADDWFC  A+B1, F
01004
01005         ENDM
01006
01007 ;*******************************************************************************
```


© 1997 Microchip Technology Inc.

```
01008
01009 ;*******************************************************************************
01010 ; NAME:          AUTONO
01011 ;
01012 ; DESCRIPTION:  Sets no auto increment or decrement
01013 ;
01014 ; TIMING (cycles): 4
01015
01016 AUTONO MACRO
01017
01018         BSF     _fs0
01019         BSF     _fs1
01020         BSF     _fs2
01021         BSF     _fs3
01022
01023         ENDM
01024
01025 ;*******************************************************************************
01026
01027 ;*******************************************************************************
01028 ; NAME:          AUTOINC
01029 ;
01030 ; DESCRIPTION:  Set auto increment
01031 ;
01032 ; TIMING (cycles): 4
01033 ;
01034
01035 AUTOINC MACRO
01036
01037         BSF     _fs0
01038         BCF     _fs1
01039         BSF     _fs2
01040         BCF     _fs3
01041
01042         ENDM
01043
01044 ;*******************************************************************************
01045
01046 ;*******************************************************************************
01047 ; NAME:          AUTODEC
01048 ;
01049 ; DESCRIPTION:  Sets auto decrement
01050 ;
01051 ; TIMING (cycles): 4
01052 ;
01053
01054 AUTODEC MACRO
```

© 1997 Microchip Technology Inc.

```
01055
01056         BCF     _fs0
01057         BCF     _fs1
01058         BCF     _fs2
01059         BCF     _fs3
01060
01061         ENDM
01062
01063 ;*****************************************************************************
01064
01065 ;*****************************************************************************
01066 ; NAME:         TFSZ32
01067 ;
01068 ; DESCRIPTION:  32 bit test and skip if zero
01069 ;
01070 ; TIMING (cycles): 6
01071 ;
01072
01073 TFSZ32 MACRO    A
01074
01075         MOVFP   A+B0,WREG
01076         IORWF   A+B1,W
01077         IORWF   A+B2,W
01078         IORWF   A+B3,W
01079         TSTFSZ  WREG
01080         ENDM
01081
01082 ;*****************************************************************************
01083
01084 ;*****************************************************************************
01085 ; NAME:         TFSZ24
01086 ;
01087 ; DESCRIPTION:  24 bit test and skip if zero
01088 ;
01089 ; TIMING (cycles): 5
01090
01091 TFSZ24 MACRO    A
01092
01093         MOVFP   A+B0,WREG
01094         IORWF   A+B1,W
01095         IORWF   A+B2,W
01096         TSTFSZ  WREG
01097         ENDM
01098
01099 ;*****************************************************************************
01100
01101 ;*****************************************************************************
```


© 1997 Microchip Technology Inc.

```
                        01102 ; NAME:         TFSZ16
                        01103 ;
                        01104 ; DESCRIPTION:  16 bit test and skip if zero
                        01105 ;
                        01106 ; TIMING (cycles): 4
                        01107 ;
                        01108
                        01109 TFSZ16 MACRO    A
                        01110
                        01111         MOVFP   A+B0,WREG
                        01112         IORWF   A+B1,W
                        01113         TSTFSZ  WREG
                        01114         ENDM
                        01115
                        01116 ;*****************************************************************************
                        00016
                        00017 ;*****************************************************************************
                        00018 ;       global  variables
                        00019 ;
                        00020         CBLOCK  0x18
00000018                00021         DPX,DPX1,DPX2,DPX3              ; arithmetic accumulator
0000001C                00022         AARG,AARG1,BARG,BARG1           ; multiply arguments
                        00023         ENDC
                        00024
                        00025         CBLOCK  0x18
00000018                00026         TMP,TMP1,TMP2,TMP3              ; temporary variables
0000001C                00027         MOVTMP,MOVTMP1,MOVTMP2,MOVTMP3  ; move temporary storage
                        00028         ENDC
                        00029
                        00030         CBLOCK  0x20
00000020                00031         VL,VL1,VL2                      ; velocity limit
00000023                00032         AL,AL1,AL2                      ; acceleration limit
                        00033
00000026                00034         KP,KP1                          ; proportional gain
00000028                00035         KV,KV1                          ; velocity gain
0000002A                00036         KI,KI1                          ; integral gain
0000002C                00037         IM                              ; integrator mode
0000002D                00038         FV,FV1                          ; velocity feedforward
0000002F                00039         FA,FA1                          ; acceleration feedforward
                        00040
00000031                00041         VALBUF,VALBUF1,VALBUF2          ; iovalue buffer
00000034                00042         CVALBUF,CVALBUF1,CVALBUF2       ; iovalue buffer
00000037                00043         DVALBUF,DVALBUF1,DVALBUF2       ; iovalue buffer
0000003A                00044         ISRBSR,ISRWREG                  ; isr save storage
0000003C                00045         CMDCHAR,CMDTEMP,CMDPTRH,CMDPTRL ; command interface variables
00000040                00046         PARTEMP,PARLEN,PARPTR           ; parameter variables
                        00047
```

© 1997 Microchip Technology Inc.

```
00000043 00048        CPOSITION,CPOSITION1,CPOSITION2          ; shutter commanded position
00000046 00049        CVELOCITY,CVELOCITY1,CVELOCITY2          ; shutter commanded velocity
00000049 00050        CMPOSITION,CMPOSITION1,CMPOSITION2       ; shutter measured position
0000004C 00051        CMVELOCITY,CMVELOCITY1,CMVELOCITY2       ; shutter measured velocity
         00052
0000004F 00053        STRVALH,STRVALL                 ; string io variables
00000051 00054        HEXVAL,HEXTMP,HEXSTAT           ; hex io variables
         00055
00000054 00056        OPOSITION,OPOSITION1
00000056 00057        OPOSITION2,OPOSITION3           ; original commanded position
00000058 00058        POSITION,POSITION1,POSITION2    ; commanded position
0000005B 00059        VELOCITY,VELOCITY1,VELOCITY2    ; commanded velocity
         00060
0000005E 00061        NMOVVAL,NMOVVAL1,NMOVVAL2,NMOVVAL3       ; move value
00000062 00062        MOVVAL,MOVVAL1,MOVVAL2,MOVVAL3           ; move value
00000066 00063        HMOVVAL,HMOVVAL1,HMOVVAL2,HMOVVAL3       ; half move value
0000006A 00064        MOVTIME,MOVTIME1                         ; move time in sample counts
         00065
0000006C 00066        MOVSIGN                         ; 0x00 for positive, 0x80 for negative
0000006D 00067        T1,T11                          ; time to maximum velocity
0000006F 00068        T2,T21                          ; time for half the move
00000071 00069        TAU,TAU1                        ; total move time
00000073 00070        NMODE                           ; next move modetype
00000074 00071        MODE                            ; move modetype
         00072
00000075 00073        MPOSITION,MPOSITION1,MPOSITION2 ; measured position
00000078 00074        MVELOCITY,MVELOCITY1
0000007A 00075        MVELOCITY2,MVELOCITY3           ; measured velocity
0000007C 00076        POSERROR,POSERROR1,POSERROR2    ; position error
0000007F 00077        VELERROR,VELERROR1,VELERROR2    ; velocity error
         00078
00000082 00079        SIGN                            ; multiply sign
         00080
00000083 00081        Y,Y1,Y2,Y3                      ; Y(k) before pwm conversion
00000087 00082        YPWM,YPWM1,YPWM2,YPWM3          ; pwm input
0000008B 00083        YPWMIN,YPWMIN1,YPWMAX,YPWMAX1   ; pwm input limits
         00084
0000008F 00085        U0,U01,U1,U11,U2,U21            ; saturated error at successive times
         00086
00000095 00087        SERVOFLAG                       ; servoflag = 0 => no servo
00000096 00088        MODETYPE                        ; mode flag(0=position,1=velocity,-1=torque)
00000097 00089        EXTSTAT                         ; external status register
00000098 00090        MOVSTAT                         ; move status register
00000099 00091        MOVFLAG                         ; move flag
0000009A 00092        SATFLAG                         ; saturation flag (1=pos,-1=neg)
0000009B 00093        INTEGRAL,INTEGRAL1              ; integrator
         00094
```


© 1997 Microchip Technology Inc.

```
0000009D  00095         C0,C01,C1,C11,C2,C21            ; difference equation coefficients
          00096
000000A3  00097         DECVAL,DECSTAT,DECTMP,DECSIGN   ; decimal io variables
          00098
000000A7  00099         A,A1,A2,A3                              ; commanded acceleration = +-AL, 0
000000AB  00100         V,V1,V2,V3                              ; commanded velocity = +-VL, 0
000000AF  00101         MOVPBUF,MOVPBUF1,MOVPBUF2,MOVPBUF3      ; commanded position buffer
000000B3  00102         MOVVBUF,MOVVBUF1,MOVVBUF2,MOVVBUF3      ; commanded velocity buffer
          00103
000000B7  00104         UPCOUNT,UPCOUNT1                ; running up counter
000000B9  00105         DOWNCOUNT,DOWNCOUNT1            ; running down counter
          00106
000000BB  00107         MOVDEL,MOVDEL1,MOVDEL2,MOVDEL3  ; move discretization delta
000000BF  00108         PH2FLAT,PH2FLAT1                ; phase 2 flat iteration counter
000000C1  00109         INDEXPOS,INDEXPOS1,INDEXPOS2    ; position at last index pulse
          00110
000000C4  00111         SHIFTNUM                        ; # of bit shifts from middle 16
          00112
          00113 ;*************************************************************************
          00114 ;       For PICMASTER Debug/servo tuning Purposes Only
          00115
000000C5  00116         CAPFLAG                         ; trace capture flag
000000C6  00117         CAPCOUNT,CAPCOUNT1              ; PICMASTER trace capture counter
000000C8  00118         CAPTMP,CAPTMP1                  ; trace capture temporary storage
          00119
          00120 ;*************************************************************************
          00121
000000CA  00122         ZERO,ONE                        ; constants
          00123
          00124         ENDC
          00125
          00126 ;*************************************************************************
          00127
          00128
0000      00129         ORG     0x0                     ; reset vector
0000 C021 00130         GOTO    Startup                 ; startup vector
          00131
0020      00132         ORG     0x20
0020 C070 00133         GOTO    InterruptPoll           ; interrupt vector
          00134
          00135
          00136 ;*************************************************************************
          00137 ; NAME:         Startup
          00138 ;
          00139 ; DESCRIPTION:  This routine is called on the hardware reset or when the
          00140 ;               program wishes to restore initial conditions. Initiali-
          00141 ;               zation of run-time constants takes place here.
```

© 1997 Microchip Technology Inc.

```
                    00142 ;
                    00143 ; RETURNS:      restart to safe and initial state
                    00144 ;
                    00145 ; STACK UTILIZATION: none
                    00146 ; TIMING (in cycles): X
                    00147
0021                00148 Startup
                    00149
0021 8406           00150         BSF     _glintd                 ; disable all interrupts
                    00151         AUTONO                          ; no auto increment or decrement
                        M
0022 8404               M         BSF     _fs0
0023 8504               M         BSF     _fs1
0024 8604               M         BSF     _fs2
0025 8704               M         BSF     _fs3
                        M
0026 B018           00152         MOVLW   0x18                    ; clear all memory locations [18,FF]
0027 4A01           00153         MOVPF   WREG,FSR0
                    00154
0028                00155 memloop
0028 2900           00156         CLRF    INDF0, F
0029 1F01           00157         INCFSZ  FSR0, F
002A C028           00158         GOTO    memloop
                    00159
002B 15CB           00160         INCF    ONE, F
                    00161
002C B803           00162         MOVLB   BANK3                   ; BANK3 initialization
002D B03F           00163         MOVLW   TCON2_INIT
002E 770A           00164         MOVFP   WREG,TCON2
                    00165
002F B07F           00166         MOVLW   PW1DCH_INIT             ; set duty cycle to midpoint
0030 720A           00167         MOVFP   WREG,PW1DCH
0031 730A           00168         MOVFP   WREG,PW2DCH
                    00169
0032 B0C0           00170         MOVLW   PW1DCL_INIT
0033 700A           00171         MOVFP   WREG,PW1DCL
0034 710A           00172         MOVFP   WREG,PW2DCL
                    00173
0035 B006           00174         MOVLW   TCON1_INIT              ; set organization of timers
0036 760A           00175         MOVFP   WREG,TCON1
                    00176
0037 B802           00177         MOVLB   BANK2                   ; BANK2 initialization
                    00178
0038 B0FF           00179         MOVLW   PR1_INIT
0039 740A           00180         MOVFP   WREG,PR1                ; initialize timer1 period
                    00181
003A B00F           00182         MOVLW   PR2_INIT
```


© 1997 Microchip Technology Inc.

```
003B 750A  00183         MOVFP   WREG,PR2              ; initialize timer2 period
           00184
           00185
003C B800  00186         MOVLB   BANK0                ; BANK0 initialization
           00187
003D B080  00188         MOVLW   T0STA_INIT
003E 650A  00189         MOVFP   WREG,RTCSTA          ; sets T0 for external input
           00190
003F B090  00191         MOVLW   RCSTA_INIT
0040 730A  00192         MOVFP   WREG,RCSTA           ; set receive status
           00193
0041 B020  00194         MOVLW   TXSTA_INIT           ; set transmit status
0042 750A  00195         MOVFP   WREG,TXSTA
           00196
0043 B019  00197         MOVLW   SPBRG_INIT           ; set baud rate
0044 770A  00198         MOVFP   WREG,SPBRG
           00199
0045 B0F3  00200         MOVLW   DDRB_INIT
0046 710A  00201         MOVFP   WREG,DDRB            ; set port B for whatever
           00202
0047 B801  00203         MOVLB   BANK1                ; BANK1 initialization
           00204
0048 B008  00205         MOVLW   0x08                 ; initialize some parameters
0049 4A27  00206         MOVPF   WREG,KP+B1           ; proportional gain
           00207
004A B050  00208         MOVLW   0x50
004B 4A29  00209         MOVPF   WREG,KV+B1           ; derivative gain
           00210
004C B004  00211         MOVLW   0x04
004D 4A2B  00212         MOVPF   WREG,KI+B1           ; integral gain
           00213
004E B001  00214         MOVLW   0x01
004F 4A24  00215         MOVPF   WREG,AL+B1           ; acceleration limit
           00216
0050 B0F0  00217         MOVLW   0xF0
0051 4A21  00218         MOVPF   WREG,VL+B1           ; velocity limit
           00219
0052 82C4  00220         BSF     SHIFTNUM,2           ; set shift number
           00221
0053 5288  00222         MOVPF   PW1DCH,YPWM+B1
           00223
0054 B080  00224         MOVLW   PWMAXL               ; initialize pwm limits
0055 4A8D  00225         MOVPF   WREG,YPWMAX+B0
0056 B0FE  00226         MOVLW   PWMAXH
0057 4A8E  00227         MOVPF   WREG,YPWMAX+B1
0058 B040  00228         MOVLW   PWMINL
0059 4A8B  00229         MOVPF   WREG,YPWMIN+B0
```

© 1997 Microchip Technology Inc.

```
005A B001          00230         MOVLW   PWMINH
005B 4A8C          00231         MOVPF   WREG,YPWMIN+B1
                   00232
005C 2916          00233         CLRF    PIR, F                  ; clear flags, set indiviual interrupts
005D 2907          00234         CLRF    INTSTA, F
005E 8517          00235         BSF     _tm2ie
005F 8307          00236         BSF     _peie
                   00237
0060 8C06          00238         BCF     _glintd                 ; enable interrupts
                   00239
0061 B802          00240         MOVLB   BANK2
                   00241
0062               00242 zeroctrs
0062 290B          00243         CLRF    TMR0L, F                ; clear up counter
0063 290C          00244         CLRF    TMR0H, F
                   00245
0064 2912          00246         CLRF    TMR3L, F                ; clear down counter
0065 2913          00247         CLRF    TMR3H, F
                   00248
0066 B0FF          00249         MOVLW   0xFF
0067 170A          00250 delay   DECFSZ  WREG, F
0068 C067          00251         GOTO    delay
                   00252
0069 6A0B          00253         MOVFP   TMR0L,WREG
006A 080C          00254         IORWF   TMR0H,W
006B 0812          00255         IORWF   TMR3L,W
006C 0813          00256         IORWF   TMR3H,W
006D 330A          00257         TSTFSZ  WREG
006E C062          00258         GOTO    zeroctrs                ; motor still moving
                   00259
006F C086          00260         GOTO    PollingLoop
                   00261
                   00262 ;*************************************************************************
                   00263
                   00264 ;*************************************************************************
                   00265 ; NAME:          InterruptPoll
                   00266
                   00267
0070               00268 InterruptPoll
                   00269
0070 4F3A          00270         MOVPF   BSR,ISRBSR              ; save BSR,WREG
0071 4A3B          00271         MOVPF   WREG,ISRWREG
                   00272
0072 B801          00273         MOVLB   BANK1
                   00274
0073 E5EC          00275         CALL    doMPosMVel              ; calculate measured position and
                   00276                                         ; velocity
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
                00277
0074 E61D       00278         CALL    doExtstat               ; evaluate external status
                00279
0075 2298       00280         RLNCF   MOVSTAT,W               ; if MOVFLAG=0 and MOVSTAT,BIT7=1
0076 B501       00281         ANDLW   0x01                    ; then do premove. This is only
0077 0499       00282         SUBWF   MOVFLAG,W               ; executed once at the beginning of
0078 9F0A       00283         BTFSC   WREG,MSB                ; each move
0079 E37E       00284         CALL    doPreMove
                00285
007A 9E98       00286         BTFSC   MOVSTAT,BIT6            ; is motion continuing?
007B E44F       00287         CALL    doMove                  ; if so, do move
                00288
007C E291       00289         CALL    doError                 ; calculate position and velocity
                00290                                         ; error
007D 3395       00291         TSTFSZ  SERVOFLAG               ; test servoflag, if 0 then no servo
007E E2D8       00292         CALL    doServo                 ; do servo
                00293
007F 33C5       00294         TSTFSZ  CAPFLAG
0080 E742       00295         CALL    doCaptureRegs           ; for PIC-MASTER Trace Capture, demo purposes
                00296
0081 B801       00297         MOVLB   BANK1
                00298
0082 2916       00299         CLRF    PIR, F                  ; clear all interrupt request flags
                00300
0083 6F3A       00301         MOVFP   ISRBSR,BSR              ; restore BSR,WREG
0084 6A3B       00302         MOVFP   ISRWREG,WREG
                00303
0085 0005       00304         RETFIE
                00305
                00306 ;*************************************************************************
                00307
                00308 ;*************************************************************************
                00309 ; NAME:                 PollingLoop
                00310 ;
                00311 ; DESCRIPTION:          The actual polling loop called after the board's
                00312 ;                       initialization
                00313 ;
                00314 ; ENTRY CONDITIONS:     System globals and hardware initialized and the
                00315 ;                       interrupt processes started.
                00316 ;
                00317
0086            00318 PollingLoop
                00319
0086 E08D       00320         CALL    IdleFunction
0087 E1AC       00321         CALL    GetChk
0088 31CB       00322         CPFSEQ  ONE                     ; GetChk, is receive buffer full?
0089 C086       00323         GOTO    PollingLoop
```

```
                00324
008A E1A1       00325          CALL    GetChar                 ; if so, get character
008B 4A3C       00326          MOVPF   WREG,CMDCHAR            ; put in CMDCHAR
008C C08F       00327          GOTO    DoCommand
                00328
                00329 ;*****************************************************************************
                00330
                00331 ;*****************************************************************************
                00332 ; NAME:          IdleFunction
                00333
                00334 ; DESCRIPTION:   This routine will perform work while doing waits in serial
                00335 ;               I/O functions.
                00336 ;
                00337
008D            00338 IdleFunction
                00339
008D 0004       00340          CLRWDT
008E 0002       00341          RETURN
                00342
                00343 ;*****************************************************************************
                00344
                00345 ;*****************************************************************************
                00346 ; NAME:          DoCommand
                00347 ;
                00348 ; DESCRIPTION:   Search command table for command and execute it.
                00349 ;
                00350
008F            00351 DoCommand
                00352
008F B066       00353          MOVLW   LOW     CMD_TABLE       ; CMD_TABLE LSB
0090 4A0D       00354          MOVPF   WREG,TBLPTRL
0091 B007       00355          MOVLW   HIGH    CMD_TABLE       ; CMD_TABLE MSB
0092 4A0E       00356          MOVPF   WREG,TBLPTRH
                00357
0093 AB3D       00358          TABLRD  1,1,CMDTEMP
0094            00359 tryNextCmd
0094 A93D       00360          TABLRD  0,1,CMDTEMP             ; read entry from table
0095 A23E       00361          TLRD    1,CMDPTRH
0096 A93F       00362          TABLRD  0,1,CMDPTRL
                00363
0097 6A3D       00364          MOVFP   CMDTEMP,WREG
0098 30CA       00365          CPFSLT  ZERO
                00366
0099 C0A4       00367          GOTO    noCommand               ; error if end of table
                00368
009A 313C       00369          CPFSEQ  CMDCHAR
009B C094       00370          GOTO    tryNextCmd
```


© 1997 Microchip Technology Inc.

```
                    00371
009C E1A4           00372          CALL    PutChar                 ; echo command
                    00373
009D 633E           00374          MOVFP   CMDPTRH,PCLATH          ; indirect jump to command routine
009E 623F           00375          MOVFP   CMDPTRL,PCL
009F 0000           00376          NOP
00A0                00377 cmdFinish
00A0 E1A4           00378          CALL    PutChar                 ; send response character from
                    00379                                          ; command routine followed by CR
00A1 B00D           00380          MOVLW   CR
00A2 E1A4           00381          CALL    PutChar
                    00382
00A3 C086           00383          GOTO    PollingLoop
                    00384
00A4                00385 noCommand
00A4 B03F           00386          MOVLW   CMD_BAD                 ; send error character
00A5 C0A0           00387          GOTO    cmdFinish
                    00388
                    00389 ;*******************************************************************************
                    00390
                    00391 ;*******************************************************************************
                    00392 ; NAME:          do_null
                    00393 ;
                    00394 ; DESCRIPTION:   The do nothing command used to determine if the chip is
                    00395 ;                working. Initiated by a carriage return.
                    00396
00A6                00397 do_null
00A6 B021           00398          MOVLW   CMD_OK
00A7 C0A0           00399          GOTO    cmdFinish
                    00400
                    00401 ;*******************************************************************************
                    00402
                    00403 ;*******************************************************************************
                    00404 ; NAME:          do_move
                    00405
                    00406 ; DESCRIPTION:   Commands the axis to move to a new position or velocity.
                    00407 ;                Position data is relative, and in encoder counts. Velocity
                    00408 ;                data is absolute, and in encoder counts/sample time multi-
                    00409 ;                plied by 256. All moves are performed by the controller such
                    00410 ;                that velocity and acceleration limits set into parameter
                    00411 ;                memory will not be violated. All move commands are kept in a
                    00412 ;                one deep FIFO buffer. The command in the buffer is executed
                    00413 ;                as soon as the currently executed command is complete.
                    00414 ;
                    00415 ;
                    00416 ; ARGUMENTS:     M [800000,7FFFFF]
                    00417 ;
```

© 1997 Microchip Technology Inc.

```
00A8                00418 do_move
                    00419
                    00420         #if     DECIO
                    00421
00A8 E217           00422         CALL    GetDecVal
                    00423
                    00424         #else
                    00425
                    00426         CALL    GetVal
                    00427
                    00428         #endif
                    00429
00A9 9F98           00430         BTFSC   MOVSTAT,BIT7            ; test if buffer available
00AA C0B4           00431         GOTO    bufoverflow
                    00432
                    00433         MOV24   VALBUF,NMOVVAL          ; if so, accept value into NMOVVAL and
                       M
00AB 6A31              M          MOVFP   VALBUF+B0,WREG          ; get byte of a into w
00AC 4A5E              M          MOVPF   WREG,NMOVVAL+B0         ; move to b(B0)
00AD 6A32              M          MOVFP   VALBUF+B1,WREG          ; get byte of a into w
00AE 4A5F              M          MOVPF   WREG,NMOVVAL+B1         ; move to b(B1)
00AF 6A33              M          MOVFP   VALBUF+B2,WREG          ; get byte of a into w
00B0 4A60              M          MOVPF   WREG,NMOVVAL+B2         ; move to b(B2)
                       M
00B1 8798           00434         BSF     MOVSTAT,BIT7            ; set buffer full flag
                    00435
00B2 B021           00436         MOVLW   CMD_OK
00B3 C0A0           00437         GOTO    cmdFinish
                    00438
00B4                00439 bufoverflow
00B4 B03F           00440         MOVLW   CMD_BAD                 ; else, return error
00B5 C0A0           00441         GOTO    cmdFinish
                    00442
                    00443 ;*************************************************************************
                    00444
                    00445 ;*************************************************************************
                    00446 ; NAME:         do_mode
                    00447 ;
                    00448 ; DESCRIPTION:  An argument of "P" will cause all subsequent move commands
                    00449 ;               to be incremental position moves. A "V" argument will cause
                    00450 ;               all subsequent moves to be absolute velocity moves.
                    00451 ;
                    00452 ; ARGUMENTS:    0 [P,V]
                    00453 ;
                    00454
00B6                00455 do_mode
                    00456
```


© 1997 Microchip Technology Inc.

```
00B6 E08D          00457          CALL    IdleFunction            ; get single character loop
00B7 E1AC          00458          CALL    GetChk
00B8 31CB          00459          CPFSEQ  ONE
00B9 C0B6          00460          GOTO    do_mode
00BA E1A1          00461          CALL    GetChar
00BB 4A50          00462          MOVPF   WREG,STRVALL
00BC 2996          00463          CLRF    MODETYPE, F             ; MODETYPE=0 for position moves
00BD               00464 testP
00BD B050          00465          MOVLW   'P'                     ; position moves for type P
00BE 3150          00466          CPFSEQ  STRVALL
00BF C0C1          00467          GOTO    testV
00C0 C0CE          00468          GOTO    modeok
00C1               00469 testV
00C1 B056          00470          MOVLW   'V'                     ; velocity moves for type V
00C2 3150          00471          CPFSEQ  STRVALL
00C3 C0C6          00472          GOTO    testT
00C4 1596          00473          INCF    MODETYPE, F             ; MODETYPE=1 for velocity moves
00C5 C0CE          00474          GOTO    modeok
00C6               00475 testT
00C6 B054          00476          MOVLW   'T'                     ; TORQUE Moves for type 'T'
00C7 3150          00477          CPFSEQ  STRVALL
00C8 C0CC          00478          GOTO    modeerror
00C9 2B96          00479          SETF    MODETYPE, F             ; MODETYPE=-1 for torque moves
00CA 2995          00480          CLRF    SERVOFLAG, F            ; disable servo
00CB C0CE          00481          GOTO    modeok
00CC               00482 modeerror
00CC B03F          00483          MOVLW   CMD_BAD                 ; mode error
00CD C0A0          00484          GOTO    cmdFinish
00CE               00485 modeok
00CE 6A50          00486          MOVFP   STRVALL,WREG            ; echo type character
00CF E1A4          00487          CALL    PutChar
                   00488
00D0 B021          00489          MOVLW   CMD_OK
00D1 C0A0          00490          GOTO    cmdFinish
                   00491
                   00492 ;*****************************************************************************
                   00493
                   00494 ;*****************************************************************************
                   00495 ; NAME:          do_setparameter
                   00496 ;
                   00497 ; DESCRIPTION:  Sets controller parameters to the value given.
                   00498 ;
                   00499 ;       Parameter               #               Range
                   00500 ;
                   00501 ;       VL=velocity limit       0               [0,7FFFFF]
                   00502 ;       AL=acceleration limit   1               [0,7FFFFF]
                   00503 ;
```

© 1997 Microchip Technology Inc.

```
                        00504 ;       KP=proportional gain    2               [8000,7FFF]
                        00505 ;       KP=velocity gain        3               [8000,7FFF]
                        00506 ;       KP=integral gain        4               [8000,7FFF]
                        00507 ;
                        00508 ;       IM=integrator mode      5               [0,3]
                        00509 ;
                        00510 ;       FV=velocity FF          6               [8000,7FFF] : Not Implemented
                        00511 ;       FA=acceleration FF      7               [8000,7FFF] : Not Implemented
                        00512 ;
                        00513 ;
                        00514 ; ARGUMENTS:    S [0,FF] [800000,7FFFFF]
                        00515 ;
                        00516
00D2                    00517 do_setparameter
                        00518
00D2 E253               00519         CALL    GetPar                          ; get parameter number
                        00520
00D3 B008               00521         MOVLW   NUMPAR                          ; check if in range [0,NUMPAR]
00D4 3031               00522         CPFSLT  VALBUF+B0
00D5 C0F7               00523         GOTO    Serror
                        00524
00D6 B089               00525         MOVLW   LOW    PAR_TABLE                ; PAR_TABLE LSB
00D7 4A0D               00526         MOVPF   WREG,TBLPTRL
00D8 B007               00527         MOVLW   HIGH   PAR_TABLE                ; PAR_TABLE MSB
00D9 4A0E               00528         MOVPF   WREG,TBLPTRH
                        00529
00DA AB40               00530         TABLRD  1,1,PARTEMP
                        00531
00DB                    00532 setNextPar
00DB A240               00533         TLRD    1,PARTEMP                       ; read entry from table
00DC A941               00534         TABLRD  0,1,PARLEN
00DD A942               00535         TABLRD  0,1,PARPTR
                        00536
00DE B008               00537         MOVLW   NUMPAR                          ; error if end of table
00DF 3040               00538         CPFSLT  PARTEMP
00E0 C0F7               00539         GOTO    Serror
                        00540
00E1 6A40               00541         MOVFP   PARTEMP,WREG
00E2 3131               00542         CPFSEQ  VALBUF+B0
00E3 C0DB               00543         GOTO    setNextPar
                        00544
00E4 6A42               00545         MOVFP   PARPTR,WREG                     ; pointer to parameter in FSR1
00E5 690A               00546         MOVFP   WREG,FSR1
                        00547
                        00548         #if     DECIO                           ; get new value in VALBUF
                        00549
00E6 E217               00550         CALL    GetDecVal
```


© 1997 Microchip Technology Inc.

```
                    00551
                    00552         #else
                    00553
                    00554         CALL    GetVal
                    00555
                    00556         #endif
                    00557
00E7 B031           00558         MOVLW   VALBUF                  ; pointer to VALBUF in FSR0
00E8 610A           00559         MOVFP   WREG,FSR0
                    00560         AUTOINC                         ; set autoincrement
                        M
00E9 8404               M         BSF     _fs0
00EA 8D04               M         BCF     _fs1
00EB 8604               M         BSF     _fs2
00EC 8F04               M         BCF     _fs3
                        M
00ED                00561 setGetMore
00ED 6800           00562         MOVFP   INDF0,INDF1             ; move new value to parameter
00EE 0741           00563         DECF    PARLEN, F
00EF 3341           00564         TSTFSZ  PARLEN
00F0 C0ED           00565         GOTO    setGetMore
                    00566
                    00567         AUTONO                          ; no autoincrement
                        M
00F1 8404               M         BSF     _fs0
00F2 8504               M         BSF     _fs1
00F3 8604               M         BSF     _fs2
00F4 8704               M         BSF     _fs3
                        M
                    00568
                    00569
00F5 B021           00570         MOVLW   CMD_OK
00F6 C0A0           00571         GOTO    cmdFinish
                    00572
00F7                00573 Serror
00F7 B03F           00574         MOVLW   CMD_BAD
00F8 C0A0           00575         GOTO    cmdFinish
                    00576
                    00577 ;*****************************************************************************
                    00578
                    00579 ;*****************************************************************************
                    00580 ; NAME:         do_readparameter
                    00581 ;
                    00582 ; DESCRIPTION:  Returns the present value of a parameter.
                    00583 ;
                    00584 ; ARGUMENTS:    R [0,FF]
                    00585 ;
```

© 1997 Microchip Technology Inc.

```
                    00586 ; RETURNS:      The present value of the requested parameter is returned.
                    00587
00F9                00588 do_readparameter
                    00589
00F9 E253           00590         CALL    GetPar                  ; get parameter number
                    00591
00FA B008           00592         MOVLW   NUMPAR                  ; check if in range [0,NUMPAR]
00FB 3031           00593         CPFSLT  VALBUF+B0
00FC C121           00594         GOTO    Rerror
                    00595
00FD B089           00596         MOVLW   LOW     PAR_TABLE       ; PAR_TABLE LSB
00FE 4A0D           00597         MOVPF   WREG,TBLPTRL
00FF B007           00598         MOVLW   HIGH    PAR_TABLE       ; PAR_TABLE MSB
0100 4A0E           00599         MOVPF   WREG,TBLPTRH
                    00600
0101 AB40           00601         TABLRD  1,1,PARTEMP
                    00602
0102                00603 readNextPar
0102 A240           00604         TLRD    1,PARTEMP               ; read entry from table
0103 A941           00605         TABLRD  0,1,PARLEN
0104 A942           00606         TABLRD  0,1,PARPTR
                    00607
0105 B008           00608         MOVLW   NUMPAR                  ; error if end of table
0106 3040           00609         CPFSLT  PARTEMP
0107 C121           00610         GOTO    Rerror
                    00611
0108 6A40           00612         MOVFP   PARTEMP,WREG
0109 3131           00613         CPFSEQ  VALBUF+B0
010A C102           00614         GOTO    readNextPar
                    00615
010B 6A42           00616         MOVFP   PARPTR,WREG             ; pointer to parameter in FSR1
010C 690A           00617         MOVFP   WREG,FSR1
                    00618
010D B031           00619         MOVLW   VALBUF                  ; pointer to VALBUF in FSR1
010E 610A           00620         MOVFP   WREG,FSR0
                    00621         AUTOINC                         ; set autoincrement
                        M
010F 8404               M         BSF     _fs0
0110 8D04               M         BCF     _fs1
0111 8604               M         BSF     _fs2
0112 8F04               M         BCF     _fs3
                        M
                    00622
                    00623         CLR24   VALBUF                  ; clear old VALBUF
0113 2931               M         CLRF    VALBUF+B0, F
0114 2932               M         CLRF    VALBUF+B1, F
0115 2933               M         CLRF    VALBUF+B2, F
```


© 1997 Microchip Technology Inc.

```
                    M
0116                00624 readGetMore
0116 6008           00625         MOVFP   INDF1,INDF0             ; read parameter into VALBUF
0117 0741           00626         DECF    PARLEN, F
0118 3341           00627         TSTFSZ  PARLEN
0119 C116           00628         GOTO    readGetMore
                    00629
                    00630         AUTONO                          ; no autoincrement
                    M
011A 8404           M             BSF     _fs0
011B 8504           M             BSF     _fs1
011C 8604           M             BSF     _fs2
011D 8704           M             BSF     _fs3
                    M
                    00631
                    00632
                    00633         #if     DECIO                   ; send parameter value
                    00634
011E E260           00635         CALL    PutDecVal
                    00636
                    00637         #else
                    00638
                    00639         CALL    PutVal
                    00640
                    00641         #endif
                    00642
011F B021           00643         MOVLW   CMD_OK
0120 C0A0           00644         GOTO    cmdFinish
                    00645
0121                00646 Rerror
0121 B03F           00647         MOVLW   CMD_BAD
0122 C0A0           00648         GOTO    cmdFinish
                    00649
                    00650 ;*****************************************************************************
                    00651
                    00652 ;*****************************************************************************
                    00653 ; NAME: do_shutter
                    00654 ;
                    00655 ; DESCRIPTION:  Returns the time (in sample time counts [0,FFFF]) since the
                    00656 ;               start of the present move and captures the commanded and
                    00657 ;               measured values of position and velocity at the time of the
                    00658 ;               command.
                    00659 ;
                    00660 ;
                    00661 ; ARGUMENTS:    C
                    00662 ;
                    00663 ; RETURNS:      The time since the start of the present move is returned.
```

© 1997 Microchip Technology Inc.

```
                    00664 ;
                    00665
0123                00666 do_shutter
                    00667
0123 8406           00668         BSF     _glintd                 ; disable all interrupts
                    00669
                    00670         MOV24   POSITION,CPOSITION      ; capture commanded position
                        M
0124 6A58               M         MOVFP   POSITION+B0,WREG        ; get byte of a into w
0125 4A43               M         MOVPF   WREG,CPOSITION+B0       ; move to b(B0)
0126 6A59               M         MOVFP   POSITION+B1,WREG        ; get byte of a into w
0127 4A44               M         MOVPF   WREG,CPOSITION+B1       ; move to b(B1)
0128 6A5A               M         MOVFP   POSITION+B2,WREG        ; get byte of a into w
0129 4A45               M         MOVPF   WREG,CPOSITION+B2       ; move to b(B2)
                        M
                    00671         MOV24   VELOCITY,CVELOCITY      ; capture commanded velocity
                        M
012A 6A5B               M         MOVFP   VELOCITY+B0,WREG        ; get byte of a into w
012B 4A46               M         MOVPF   WREG,CVELOCITY+B0       ; move to b(B0)
012C 6A5C               M         MOVFP   VELOCITY+B1,WREG        ; get byte of a into w
012D 4A47               M         MOVPF   WREG,CVELOCITY+B1       ; move to b(B1)
012E 6A5D               M         MOVFP   VELOCITY+B2,WREG        ; get byte of a into w
012F 4A48               M         MOVPF   WREG,CVELOCITY+B2       ; move to b(B2)
                        M
                    00672         MOV24   MPOSITION,CMPOSITION    ; capture measured position
                        M
0130 6A75               M         MOVFP   MPOSITION+B0,WREG       ; get byte of a into w
0131 4A49               M         MOVPF   WREG,CMPOSITION+B0      ; move to b(B0)
0132 6A76               M         MOVFP   MPOSITION+B1,WREG       ; get byte of a into w
0133 4A4A               M         MOVPF   WREG,CMPOSITION+B1      ; move to b(B1)
0134 6A77               M         MOVFP   MPOSITION+B2,WREG       ; get byte of a into w
0135 4A4B               M         MOVPF   WREG,CMPOSITION+B2      ; move to b(B2)
                        M
                    00673         MOV24   MVELOCITY,CMVELOCITY    ; capture measured velocity
                        M
0136 6A78               M         MOVFP   MVELOCITY+B0,WREG       ; get byte of a into w
0137 4A4C               M         MOVPF   WREG,CMVELOCITY+B0      ; move to b(B0)
0138 6A79               M         MOVFP   MVELOCITY+B1,WREG       ; get byte of a into w
0139 4A4D               M         MOVPF   WREG,CMVELOCITY+B1      ; move to b(B1)
013A 6A7A               M         MOVFP   MVELOCITY+B2,WREG       ; get byte of a into w
013B 4A4E               M         MOVPF   WREG,CMVELOCITY+B2      ; move to b(B2)
                        M
                    00674
013C 2933           00675         CLRF    VALBUF+B2, F
                    00676         MOV16   MOVTIME,VALBUF          ; capture move time, move to VALBUF
                        M
013D 6A6A               M         MOVFP   MOVTIME+B0,WREG         ; get byte of a into w
```


© 1997 Microchip Technology Inc.

```
013E 4A31              M        MOVPF   WREG,VALBUF+B0          ; move to b(B0)
013F 6A6B              M        MOVFP   MOVTIME+B1,WREG         ; get byte of a into w
0140 4A32              M        MOVPF   WREG,VALBUF+B1          ; move to b(B1)
                       M
                    00677
0141 8C06           00678        BCF     _glintd                 ; enable all interrupts
                    00679
                    00680        #if     DECIO
                    00681
0142 E260           00682        CALL    PutDecVal
                    00683
                    00684        #else
                    00685
                    00686        CALL    PutVal
                    00687
                    00688        #endif
                    00689
0143 B021           00690        MOVLW   CMD_OK
0144 C0A0           00691        GOTO    cmdFinish
                    00692
                    00693 ;*****************************************************************************
                    00694
                    00695 ;*****************************************************************************
                    00696 ; NAME: do_readcomposition
                    00697 ;
                    00698 ; DESCRIPTION:  Returns the commanded position count which was captured
                    00699 ;               during the last shutter command.
                    00700 ;
                    00701 ; ARGUMENTS:    P
                    00702 ;
                    00703 ; RETURNS:      The last captured position count is returned. [800000,7FFFFF]
                    00704 ;
                    00705
0145                00706 do_readcomposition
                    00707
                    00708        MOV24   CPOSITION,VALBUF        ; move CPOSITION to VALBUF
                       M
0145 6A43              M        MOVFP   CPOSITION+B0,WREG       ; get byte of a into w
0146 4A31              M        MOVPF   WREG,VALBUF+B0          ; move to b(B0)
0147 6A44              M        MOVFP   CPOSITION+B1,WREG       ; get byte of a into w
0148 4A32              M        MOVPF   WREG,VALBUF+B1          ; move to b(B1)
0149 6A45              M        MOVFP   CPOSITION+B2,WREG       ; get byte of a into w
014A 4A33              M        MOVPF   WREG,VALBUF+B2          ; move to b(B2)
                       M
                    00709
                    00710        #if     DECIO
                    00711
```

© 1997 Microchip Technology Inc.

```
014B E260          00712         CALL    PutDecVal
                   00713
                   00714         #else
                   00715
                   00716         CALL    PutVal
                   00717
                   00718         #endif
                   00719
014C B021          00720         MOVLW   CMD_OK
014D C0A0          00721         GOTO    cmdFinish
                   00722
                   00723 ;*****************************************************************************
                   00724
                   00725 ;*****************************************************************************
                   00726 ; NAME: do_readcomvelocity
                   00727 ;
                   00728 ; DESCRIPTION:  Returns the commanded velocity multiplied by 256 which was
                   00729 ;               captured during the last shutter command.
                   00730 ;
                   00731 ; ARGUMENTS:    V
                   00732 ;
                   00733 ; RETURNS:      The last captured commanded velocity times 256 is returned.
                   00734 ;               [800000,7FFFFF]
                   00735 ;
                   00736
014E               00737 do_readcomvelocity
                   00738
                   00739         MOV24   CVELOCITY,VALBUF        ; move commanded velocity to VALBUF
                      M
014E 6A46             M          MOVFP   CVELOCITY+B0,WREG       ; get byte of a into w
014F 4A31             M          MOVPF   WREG,VALBUF+B0          ; move to b(B0)
0150 6A47             M          MOVFP   CVELOCITY+B1,WREG       ; get byte of a into w
0151 4A32             M          MOVPF   WREG,VALBUF+B1          ; move to b(B1)
0152 6A48             M          MOVFP   CVELOCITY+B2,WREG       ; get byte of a into w
0153 4A33             M          MOVPF   WREG,VALBUF+B2          ; move to b(B2)
                      M
                   00740
                   00741         #if     DECIO
                   00742
0154 E260          00743         CALL    PutDecVal
                   00744
                   00745         #else
                   00746
                   00747         CALL    PutVal
                   00748
                   00749         #endif
                   00750
```


© 1997 Microchip Technology Inc.

```
0155 B021          00751         MOVLW   CMD_OK
0156 C0A0          00752         GOTO    cmdFinish
                   00753
                   00754 ;*****************************************************************************
                   00755
                   00756 ;*****************************************************************************
                   00757 ; NAME: do_readactposition
                   00758 ;
                   00759 ; DESCRIPTION:  Returns the measured position count which was captured during
                   00760 ;               the last shutter command.
                   00761 ;
                   00762 ; ARGUMENTS:    p
                   00763 ;
                   00764 ; RETURNS:      The last captured measured position count is returned.
                   00765 ;               [800000,7FFFFF]
                   00766 ;
                   00767
0157               00768 do_readactposition
                   00769
                   00770         MOV24   CMPOSITION,VALBUF       ; move measured position to VALBUF
                       M
0157 6A49              M         MOVFP   CMPOSITION+B0,WREG      ; get byte of a into w
0158 4A31              M         MOVPF   WREG,VALBUF+B0          ; move to b(B0)
0159 6A4A              M         MOVFP   CMPOSITION+B1,WREG      ; get byte of a into w
015A 4A32              M         MOVPF   WREG,VALBUF+B1          ; move to b(B1)
015B 6A4B              M         MOVFP   CMPOSITION+B2,WREG      ; get byte of a into w
015C 4A33              M         MOVPF   WREG,VALBUF+B2          ; move to b(B2)
                       M
                   00771
                   00772         #if     DECIO
                   00773
015D E260          00774         CALL    PutDecVal
                   00775
                   00776         #else
                   00777
                   00778         CALL    PutVal
                   00779
                   00780         #endif
                   00781
015E B021          00782         MOVLW   CMD_OK
015F C0A0          00783         GOTO    cmdFinish
                   00784
                   00785 ;*****************************************************************************
                   00786
                   00787 ;*****************************************************************************
                   00788 ; NAME: do_readactvelocity
                   00789 ;
```

© 1997 Microchip Technology Inc.

```
                    00790 ; DESCRIPTION:  Returns the measured velocity multiplied by 256 which was
                    00791 ;               captured during the last shutter command.
                    00792 ;
                    00793 ; ARGUMENTS:    v
                    00794 ;
                    00795 ; RETURNS:      The last captured measured velocity times 256 is returned.
                    00796 ;               [800000,7FFFFF]
                    00797 ;
                    00798
0160                00799 do_readactvelocity
                    00800
                    00801         MOV24   CMVELOCITY,VALBUF       ; move measured velocity to VALBUF
                        M
0160 6A4C           M             MOVFP   CMVELOCITY+B0,WREG      ; get byte of a into w
0161 4A31           M             MOVPF   WREG,VALBUF+B0          ; move to b(B0)
0162 6A4D           M             MOVFP   CMVELOCITY+B1,WREG      ; get byte of a into w
0163 4A32           M             MOVPF   WREG,VALBUF+B1          ; move to b(B1)
0164 6A4E           M             MOVFP   CMVELOCITY+B2,WREG      ; get byte of a into w
0165 4A33           M             MOVPF   WREG,VALBUF+B2          ; move to b(B2)
                    M
                    00802
                    00803         #if     DECIO
                    00804
0166 E260           00805         CALL    PutDecVal
                    00806
                    00807         #else
                    00808
                    00809         CALL    PutVal
                    00810
                    00811         #endif
                    00812
0167 B021           00813         MOVLW   CMD_OK
0168 C0A0           00814         GOTO    cmdFinish
                    00815
                    00816 ;*****************************************************************************
                    00817
                    00818 ;*****************************************************************************
                    00819 ; NAME: do_externalstatus
                    00820 ;
                    00821 ; DESCRIPTION:  Returns a two digit hex number which defines the state of
                    00822 ;               the bits in the external status register. Issuing this
                    00823 ;               command will clear all the bits in the external status
                    00824 ;               register unless the event which set the bit is still true.
                    00825 ;
                    00826 ;
                    00827 ; ARGUMENTS:    X
                    00828 ;
```


© 1997 Microchip Technology Inc.

```
                        00829 ; RETURNS:      The external status register is returned.
                        00830 ;
                        00831
0169                    00832 do_externalstatus
                        00833
0169 8406               00834         BSF     _glintd
016A 6A97               00835         MOVFP   EXTSTAT,WREG
016B 2997               00836         CLRF    EXTSTAT, F
016C 8C06               00837         BCF     _glintd
016D E1B3               00838         CALL    PutHex
                        00839
016E B021               00840         MOVLW   CMD_OK
016F C0A0               00841         GOTO    cmdFinish
                        00842
                        00843 ;*****************************************************************************
                        00844
                        00845 ;*****************************************************************************
                        00846 ; NAME: do_movestatus
                        00847 ;
                        00848 ; DESCRIPTION:  Returns a two digit hex number which defines the state of
                        00849 ;               the bits in the move status register. Issuing this command
                        00850 ;               will clear all the bits in the move status register unless
                        00851 ;               the event which set the bit is still true.
                        00852 ;
                        00853 ; ARGUMENTS:    Y
                        00854 ;
                        00855 ; RETURNS:      The move status register is returned.
                        00856 ;
                        00857
0170                    00858 do_movestatus
                        00859
0170 6A98               00860         MOVFP   MOVSTAT,WREG
0171 E1B3               00861         CALL    PutHex
                        00862
0172 B021               00863         MOVLW   CMD_OK
0173 C0A0               00864         GOTO    cmdFinish
                        00865
                        00866 ;*****************************************************************************
                        00867
                        00868 ;*****************************************************************************
                        00869 ; NAME:         do_readindposition
                        00870 ;
                        00871 ; DESCRIPTION:  Returns the last index position captured in position counts.
                        00872 ;
                        00873 ; ARGUMENTS:    I
                        00874 ;
                        00875 ; RETURNS:      The last captured index position is returned.
```

© 1997 Microchip Technology Inc.

```
                    00876 ;
                    00877
0174                00878 do_readindposition
                    00879
                    00880           MOV24   INDEXPOS,VALBUF         ; move measured velocity to VALBUF
                        M
0174 6AC1           M               MOVFP   INDEXPOS+B0,WREG        ; get byte of a into w
0175 4A31           M               MOVPF   WREG,VALBUF+B0          ; move to b(B0)
0176 6AC2           M               MOVFP   INDEXPOS+B1,WREG        ; get byte of a into w
0177 4A32           M               MOVPF   WREG,VALBUF+B1          ; move to b(B1)
0178 6AC3           M               MOVFP   INDEXPOS+B2,WREG        ; get byte of a into w
0179 4A33           M               MOVPF   WREG,VALBUF+B2          ; move to b(B2)
                    M
                    00881
                    00882           #if     DECIO
                    00883
017A E260           00884           CALL    PutDecVal
                    00885
                    00886           #else
                    00887
                    00888           CALL    PutVal
                    00889
                    00890           #endif
                    00891
017B B021           00892           MOVLW   CMD_OK
017C C0A0           00893           GOTO    cmdFinish
                    00894
                    00895 ;*****************************************************************************
                    00896
                    00897 ;*****************************************************************************
                    00898 ; NAME:           do_setposition
                    00899 ;
                    00900 ; DESCRIPTION:    Sets the measured and commanded position to the value given.
                    00901 ;                 This command should not be sent unless the move FIFO buffer is empty.
                    00902 ;
                    00903 ; ARGUMENTS:      H [800000,7FFFFF]
                    00904 ;
                    00905
017D                00906 do_setposition
                    00907
                    00908           #if     DECIO
                    00909
017D E217           00910           CALL    GetDecVal
                    00911
                    00912           #else
                    00913
                    00914           CALL    GetVal
```


© 1997 Microchip Technology Inc.

```
                    00915
                    00916         #endif
                    00917
                    00918         MOV24   VALBUF,POSITION
                        M
017E 6A31               M         MOVFP   VALBUF+B0,WREG          ; get byte of a into w
017F 4A58               M         MOVPF   WREG,POSITION+B0        ; move to b(B0)
0180 6A32               M         MOVFP   VALBUF+B1,WREG          ; get byte of a into w
0181 4A59               M         MOVPF   WREG,POSITION+B1        ; move to b(B1)
0182 6A33               M         MOVFP   VALBUF+B2,WREG          ; get byte of a into w
0183 4A5A               M         MOVPF   WREG,POSITION+B2        ; move to b(B2)
                        M
                    00919         MOV24   VALBUF,MPOSITION
                        M
0184 6A31               M         MOVFP   VALBUF+B0,WREG          ; get byte of a into w
0185 4A75               M         MOVPF   WREG,MPOSITION+B0       ; move to b(B0)
0186 6A32               M         MOVFP   VALBUF+B1,WREG          ; get byte of a into w
0187 4A76               M         MOVPF   WREG,MPOSITION+B1       ; move to b(B1)
0188 6A33               M         MOVFP   VALBUF+B2,WREG          ; get byte of a into w
0189 4A77               M         MOVPF   WREG,MPOSITION+B2       ; move to b(B2)
                        M
                    00920
                    00921         CLR32   Y
018A 2983               M         CLRF    Y+B0, F
018B 2984               M         CLRF    Y+B1, F
018C 2985               M         CLRF    Y+B2, F
018D 2986               M         CLRF    Y+B3, F
                        M
                    00922
018E B021           00923         MOVLW   CMD_OK
018F C0A0           00924         GOTO    cmdFinish
                    00925
                    00926 ;*************************************************************************
                    00927
                    00928 ;*************************************************************************
                    00929 ; NAME:         do_reset
                    00930 ;
                    00931 ; DESCRIPTION:  Performs a software reset.
                    00932 ;
                    00933 ; ARGUMENTS:    Z
                    00934 ;
                    00935
0190                00936 do_reset
                    00937
0190 B021           00938         MOVLW   CMD_OK
0191 E1A4           00939         CALL    PutChar
0192 C021           00940         GOTO    Startup
```

© 1997 Microchip Technology Inc.

```
                    00941
                    00942
                    00943 ;*************************************************************************
                    00944 ; NAME:         do_stop
                    00945 ;
                    00946 ; DESCRIPTION:  Stops servo by clearing SERVOFLAG.
                    00947 ;
0193                00948 do_stop
                    00949
0193 2995           00950         CLRF    SERVOFLAG, F
                    00951
0194 B021           00952         MOVLW   CMD_OK
0195 C0A0           00953         GOTO    cmdFinish
                    00954
                    00955 ;*************************************************************************
                    00956
                    00957 ;*************************************************************************
                    00958 ; NAME:         do_capture
                    00959 ;
                    00960
0196                00961 do_capture
                    00962
                    00963         #if     DECIO
                    00964
0196 E217           00965         CALL    GetDecVal
                    00966
                    00967         #else
                    00968
                    00969         CALL    GetVal
                    00970
                    00971         #endif
                    00972
                    00973         MOV16   VALBUF,CAPCOUNT
                        M
0197 6A31               M         MOVFP   VALBUF+B0,WREG          ; get byte of a into w
0198 4AC6               M         MOVPF   WREG,CAPCOUNT+B0        ; move to b(B0)
0199 6A32               M         MOVFP   VALBUF+B1,WREG          ; get byte of a into w
019A 4AC7               M         MOVPF   WREG,CAPCOUNT+B1        ; move to b(B1)
                        M
                    00974         MOV16   VALBUF,CAPTMP
                        M
019B 6A31               M         MOVFP   VALBUF+B0,WREG          ; get byte of a into w
019C 4AC8               M         MOVPF   WREG,CAPTMP+B0          ; move to b(B0)
019D 6A32               M         MOVFP   VALBUF+B1,WREG          ; get byte of a into w
019E 4AC9               M         MOVPF   WREG,CAPTMP+B1          ; move to b(B1)
                        M
                    00975
```


© 1997 Microchip Technology Inc.

```
019F B021          00976         MOVLW   CMD_OK
01A0 C0A0          00977         GOTO    cmdFinish
                   00978
                   00979 ;*****************************************************************************
                   00980 ; NAME:          GetChar
                   00981 ;
                   00982 ; DESCRIPTION:   Get character from receive buffer.
                   00983 ;
01A1               00984 GetChar
                   00985
01A1 B800          00986         MOVLB   BANK0                   ; set bank0
01A2 540A          00987         MOVPF   RCREG,WREG              ; receive character
                   00988
01A3 0002          00989         RETURN
                   00990
                   00991 ;*****************************************************************************
                   00992
                   00993 ;*****************************************************************************
                   00994 ; NAME:          PutChar
                   00995 ;
                   00996 ; DESCRIPTION:   send character out the serial port
                   00997 ;
                   00998 ; ARGUMENTS:     WREG contains byte to be transmitted
                   00999 ;
                   01000
01A4               01001 PutChar
                   01002
01A4 B801          01003         MOVLB   BANK1                   ; set bank1
01A5               01004 bufwait                                 ; is transmit buffer empty?
01A5 9116          01005         BTFSS   _tbmt
01A6 C1A5          01006         GOTO    bufwait
                   01007
01A7 B800          01008         MOVLB   BANK0                   ; set bank0
01A8               01009 shfwait
01A8 9115          01010         BTFSS   _trmt                   ; is transmit shift register empty?
01A9 C1A8          01011         GOTO    shfwait
                   01012
01AA 4A16          01013         MOVPF   WREG,TXREG              ; if so, send character
                   01014
01AB 0002          01015         RETURN
                   01016
                   01017 ;*****************************************************************************
                   01018
                   01019 ;*****************************************************************************
                   01020 ; NAME:          GetChk
                   01021 ;
                   01022 ; DESCRIPTION:   Check if character is in receive buffer.
```

© 1997 Microchip Technology Inc.

```
                    01023 ;
                    01024
01AC                01025 GetChk
01AC B801           01026         MOVLB   BANK1                   ; set bank1
01AD 560A           01027         MOVPF   PIR,WREG
01AE B501           01028         ANDLW   CHARREADY               ; return status in WREG
01AF 0002           01029         RETURN
                    01030
                    01031 ;*****************************************************************************
                    01032
                    01033 ;*****************************************************************************
                    01034 ; NAME: PutDec
                    01035 ;
                    01036 ; DESCRIPTION:  Converts a hex value [0,F] in WREG to its ASCII equivalent.
                    01037 ;               The upper nibble of WREG is assumed to be zero.
                    01038 ;
                    01039 ; ENTRY CONDITIONS: WREG = value to be converted and sent in ASCII decimal
                    01040 ;
                    01041
                    01042         #if     DECIO
                    01043
01B0                01044 PutDec
01B0 B130           01045         ADDLW   0x30                    ; convert to ASCII
01B1 E1A4           01046         CALL    PutChar
01B2 0002           01047         RETURN
                    01048
                    01049         #endif
                    01050
                    01051 ;*****************************************************************************
                    01052
                    01053 ;*****************************************************************************
                    01054 ; NAME: PutHex
                    01055 ;
                    01056 ; DESCRIPTION:  Convert the WREG value to ASCII hexadecimal. The output
                    01057 ;               format is two digits with the A-F parts in upper case and
                    01058 ;               leading zeros. The result is sent out the serial port with
                    01059 ;               PutChar.
                    01060 ;
                    01061 ; ENTRY CONDITIONS: WREG = value to be converted and sent in ASCII hex
                    01062 ;
                    01063
01B3                01064 PutHex
                    01065
01B3 4A51           01066         MOVPF   WREG,HEXVAL
01B4 1D0A           01067         SWAPF   WREG, F
01B5 B50F           01068         ANDLW   0x0F
01B6 4A52           01069         MOVPF   WREG,HEXTMP
```


© 1997 Microchip Technology Inc.

```
01B7 2D0A          01070          NEGW    WREG, F
01B8 B109          01071          ADDLW   0x09
01B9 970A          01072          BTFSS   WREG,MSB
01BA C1BE          01073          GOTO    puth20
01BB B037          01074          MOVLW   'A'-0x0A
01BC 0E52          01075          ADDWF   HEXTMP,W
01BD C1C0          01076          GOTO    puth25
01BE               01077 puth20
01BE B030          01078          MOVLW   '0'
01BF 0E52          01079          ADDWF   HEXTMP,W
01C0               01080 puth25
01C0 E1A4          01081          CALL    PutChar
                   01082
01C1 6A51          01083          MOVFP   HEXVAL,WREG
01C2 B50F          01084          ANDLW   0x0F
01C3 4A52          01085          MOVPF   WREG,HEXTMP
01C4 2D0A          01086          NEGW    WREG, F
01C5 B109          01087          ADDLW   0x09
01C6 970A          01088          BTFSS   WREG,MSB
01C7 C1CB          01089          GOTO    putl20
01C8 B037          01090          MOVLW   'A'-0x0A
01C9 0E52          01091          ADDWF   HEXTMP,W
01CA C1CD          01092          GOTO    putl25
01CB               01093 putl20
01CB B030          01094          MOVLW   '0'
01CC 0E52          01095          ADDWF   HEXTMP,W
01CD               01096 putl25
01CD E1A4          01097          CALL    PutChar
                   01098
01CE 0002          01099          RETURN
                   01100
                   01101 ;*************************************************************************
                   01102
                   01103 ;*************************************************************************
                   01104 ; NAME:          PutStr
                   01105 ;
                   01106 ; DESCRIPTION:   Sends a character string out the serial port.
                   01107 ;
                   01108
01CF               01109 PutStr
01CF AB4F          01110          TABLRD  1,1,STRVALH
01D0               01111 GetNextPair
01D0 A24F          01112          TLRD    1,STRVALH
01D1 A950          01113          TABLRD  0,1,STRVALL
                   01114
01D2 6A4F          01115          MOVFP   STRVALH,WREG
01D3 31CA          01116          CPFSEQ  ZERO
```

© 1997 Microchip Technology Inc.

```
01D4 C1D6          01117         GOTO    putH
01D5 0002          01118         RETURN
01D6               01119 putH
01D6 E1A4          01120         CALL    PutChar
                   01121
01D7 6A50          01122         MOVFP   STRVALL,WREG
01D8 31CA          01123         CPFSEQ  ZERO
01D9 C1DB          01124         GOTO    putL
01DA 0002          01125         RETURN
01DB               01126 putL
01DB E1A4          01127         CALL    PutChar
                   01128
01DC C1D0          01129         GOTO    GetNextPair
                   01130
                   01131 ;*****************************************************************************
                   01132
                   01133 ;*****************************************************************************
                   01134 ; NAME:         GetHex
                   01135 ;
                   01136 ; DESCRIPTION:  Receive an ASCII hex character from the serial port and
                   01137 ;               convert to numerical value.
                   01138 ;
                   01139 ; RETURNS:      numerical value in HEXVAL
                   01140
01DD               01141 GetHex
                   01142
01DD               01143 getnxt
01DD E08D          01144         CALL    IdleFunction
01DE E1AC          01145         CALL    GetChk
01DF 31CB          01146         CPFSEQ  ONE
01E0 C1DD          01147         GOTO    getnxt
                   01148
01E1 2953          01149         CLRF    HEXSTAT, F
01E2 E1A1          01150         CALL    GetChar
01E3 4A51          01151         MOVPF   WREG,HEXVAL
01E4 E1A4          01152         CALL    PutChar
01E5 B00D          01153         MOVLW   CR
01E6 0451          01154         SUBWF   HEXVAL,W
01E7 330A          01155         TSTFSZ  WREG
01E8 C1EA          01156         GOTO    gth10
01E9 C1F4          01157         GOTO    gthCR
01EA               01158 gth10
                   01159
01EA 6A51          01160         MOVFP   HEXVAL,WREG
01EB B239          01161         SUBLW   '9'
01EC 970A          01162         BTFSS   WREG,MSB
01ED C1F0          01163         GOTO    gth20
```


© 1997 Microchip Technology Inc.

```
                    01164
01EE B009           01165           MOVLW   0x09
01EF 0F51           01166           ADDWF   HEXVAL, F
                    01167
01F0                01168 gth20
01F0 B00F           01169           MOVLW   0x0F
01F1 0B51           01170           ANDWF   HEXVAL, F
01F2 2953           01171           CLRF    HEXSTAT, F
01F3 0002           01172           RETURN
                    01173
01F4                01174 gthCR
01F4 B001           01175           MOVLW   0x01
01F5 4A53           01176           MOVPF   WREG,HEXSTAT
01F6 0002           01177           RETURN
                    01178
                    01179 ;*************************************************************************
                    01180
                    01181 ;*************************************************************************
                    01182 ; NAME:           GetDec
                    01183 ;
                    01184 ; DESCRIPTION:    Receive an ASCII decimal character from the serial port and
                    01185 ;                 convert to its numerical value.
                    01186 ;
                    01187 ; ARGUMENTS:      numerical value is returned in DECVAL
                    01188 ;
                    01189
                    01190           #if     DECIO
                    01191
01F7                01192 GetDec
                    01193
01F7                01194 getdecnxt
01F7 E08D           01195           CALL    IdleFunction
01F8 E1AC           01196           CALL    GetChk
01F9 31CB           01197           CPFSEQ  ONE
01FA C1F7           01198           GOTO    getdecnxt
                    01199
01FB E1A1           01200           CALL    GetChar
01FC 4AA3           01201           MOVPF   WREG,DECVAL
01FD E1A4           01202           CALL    PutChar
                    01203
01FE B00D           01204           MOVLW   CR
01FF 04A3           01205           SUBWF   DECVAL,W
0200 30CA           01206           CPFSLT  ZERO
0201 C20E           01207           GOTO    gtdCR
0202 B02D           01208           MOVLW   MN
0203 04A3           01209           SUBWF   DECVAL,W
0204 30CA           01210           CPFSLT  ZERO
```

© 1997 Microchip Technology Inc.

```
0205 C211          01211          GOTO    gtdMN
0206 B020          01212          MOVLW   SP
0207 04A3          01213          SUBWF   DECVAL,W
0208 30CA          01214          CPFSLT  ZERO
0209 C214          01215          GOTO    gtdSP
020A               01216 gtd09
020A B00F          01217          MOVLW   0x0F
020B 0BA3          01218          ANDWF   DECVAL, F
020C 29A4          01219          CLRF    DECSTAT, F
020D 0002          01220          RETURN
020E               01221 gtdCR
020E B002          01222          MOVLW   DEC_CR
020F 4AA4          01223          MOVPF   WREG,DECSTAT
0210 0002          01224          RETURN
0211               01225 gtdMN
0211 B001          01226          MOVLW   DEC_MN
0212 4AA4          01227          MOVPF   WREG,DECSTAT
0213 0002          01228          RETURN
0214               01229 gtdSP
0214 B000          01230          MOVLW   DEC_SP
0215 4AA4          01231          MOVPF   WREG,DECSTAT
0216 0002          01232          RETURN
                   01233
                   01234          #endif
                   01235
                   01236 ;*********************************************************************************
                   01237
                   01238 ;*********************************************************************************
                   01239 ; NAME:          getval
                   01240 ;
                   01241 ; DESCRIPTION:   Get a value [800000,7FFFFF] from the serial port and place
                   01242 ;                it in VALBUF.
                   01243 ;
                   01244          #if     DECIO
                   01245          #else
                   01246
                   01247 GetVal
                   01248          CLR24   VALBUF
                   01249 getnext
                   01250          CALL    GetHex
                   01251
                   01252          MOVLW   0x01
                   01253          CPFSEQ  HEXSTAT
                   01254          GOTO    shift
                   01255          RETURN
                   01256 shift
                   01257          SWAPF   VALBUF+B2
```


© 1997 Microchip Technology Inc.

```
                01258         MOVFP   VALBUF+B2,WREG
                01259         ANDLW   0xF0
                01260         MOVPF   WREG,VALBUF+B2
                01261         SWAPF   VALBUF+B1
                01262         MOVFP   VALBUF+B1,WREG
                01263         ANDLW   0x0F
                01264         ADDWF   VALBUF+B2, F
                01265         MOVFP   VALBUF+B1,WREG
                01266         ANDLW   0xF0
                01267         MOVPF   WREG,VALBUF+B1
                01268         SWAPF   VALBUF+B0
                01269         MOVFP   VALBUF+B0,WREG
                01270         ANDLW   0x0F
                01271         ADDWF   VALBUF+B1
                01272         MOVFP   VALBUF+B0,WREG
                01273         ANDLW   0xF0
                01274         ADDWF   HEXVAL,W
                01275         MOVPF   WREG,VALBUF+B0
                01276
                01277         GOTO    getnext
                01278
                01279         #endif
                01280
                01281 ;*****************************************************************************
                01282
                01283 ;*****************************************************************************
                01284 ; NAME:         GetDecVal
                01285 ;
                01286 ; DESCRIPTION:  Get a value [-8388608,8388607] from the serial port and
                01287 ;               place it in VALBUF
                01288 ;
                01289 ; RETURNS:      numerical value is returned in VALBUF
                01290
                01291         #if     DECIO
                01292
0217            01293 GetDecVal
                01294         CLR24   VALBUF
0217 2931           M         CLRF    VALBUF+B0, F
0218 2932           M         CLRF    VALBUF+B1, F
0219 2933           M         CLRF    VALBUF+B2, F
                    M
021A E1F7       01295         CALL    GetDec
021B 2BA6       01296         SETF    DECSIGN, F
021C B001       01297         MOVLW   DEC_MN
021D 31A4       01298         CPFSEQ  DECSTAT
021E 29A6       01299         CLRF    DECSIGN, F
                01300
```

© 1997 Microchip Technology Inc.

```
021F                01301 getdecnext
021F E1F7           01302         CALL    GetDec
                    01303
0220 B002           01304         MOVLW   DEC_CR
0221 31A4           01305         CPFSEQ  DECSTAT
0222 C224           01306         GOTO    mul10
0223 C248           01307         GOTO    fixsign
0224                01308 mul10
                    01309
                    01310         RLC24   VALBUF                  ; multiply VALBUF by two
                        M
0224 8804               M         BCF     _carry
0225 1B31               M         RLCF    VALBUF+B0, F
0226 1B32               M         RLCF    VALBUF+B1, F
0227 1B33               M         RLCF    VALBUF+B2, F
                        M
                    01311         MOV24   VALBUF,DVALBUF          ; save in DVALBUF
                        M
0228 6A31               M         MOVFP   VALBUF+B0,WREG          ; get byte of a into w
0229 4A37               M         MOVPF   WREG,DVALBUF+B0         ; move to b(B0)
022A 6A32               M         MOVFP   VALBUF+B1,WREG          ; get byte of a into w
022B 4A38               M         MOVPF   WREG,DVALBUF+B1         ; move to b(B1)
022C 6A33               M         MOVFP   VALBUF+B2,WREG          ; get byte of a into w
022D 4A39               M         MOVPF   WREG,DVALBUF+B2         ; move to b(B2)
                        M
                    01312         RLC24   VALBUF
                        M
022E 8804               M         BCF     _carry
022F 1B31               M         RLCF    VALBUF+B0, F
0230 1B32               M         RLCF    VALBUF+B1, F
0231 1B33               M         RLCF    VALBUF+B2, F
                        M
                    01313         RLC24   VALBUF                  ; VALBUF now multiplied by eight
                        M
0232 8804               M         BCF     _carry
0233 1B31               M         RLCF    VALBUF+B0, F
0234 1B32               M         RLCF    VALBUF+B1, F
0235 1B33               M         RLCF    VALBUF+B2, F
                        M
                    01314         ADD24   DVALBUF,VALBUF          ; VALBUF now multiplied by ten
                        M
0236 6A37               M         MOVFP   DVALBUF+B0,WREG         ; get lowest byte of a into w
0237 0F31               M         ADDWF   VALBUF+B0, F            ; add lowest byte of b, save in b(B0)
0238 6A38               M         MOVFP   DVALBUF+B1,WREG         ; get 2nd byte of a into w
0239 1132               M         ADDWFC  VALBUF+B1, F            ; add 2nd byte of b, save in b(B1)
023A 6A39               M         MOVFP   DVALBUF+B2,WREG         ; get 3rd byte of a into w
023B 1133               M         ADDWFC  VALBUF+B2, F            ; add 3rd byte of b, save in b(B2)
```


© 1997 Microchip Technology Inc.

```
                    M
                 01315          CLR24   DVALBUF
023C 2937           M          CLRF    DVALBUF+B0, F
023D 2938           M          CLRF    DVALBUF+B1, F
023E 2939           M          CLRF    DVALBUF+B2, F
                    M
023F 6AA3        01316          MOVFP   DECVAL,WREG
0240 4A37        01317          MOVPF   WREG,DVALBUF+B0
                 01318          ADD24   DVALBUF,VALBUF
                    M
0241 6A37           M          MOVFP   DVALBUF+B0,WREG         ; get lowest byte of a into w
0242 0F31           M          ADDWF   VALBUF+B0, F            ; add lowest byte of b, save in b(B0)
0243 6A38           M          MOVFP   DVALBUF+B1,WREG         ; get 2nd byte of a into w
0244 1132           M          ADDWFC  VALBUF+B1, F            ; add 2nd byte of b, save in b(B1)
0245 6A39           M          MOVFP   DVALBUF+B2,WREG         ; get 3rd byte of a into w
0246 1133           M          ADDWFC  VALBUF+B2, F            ; add 3rd byte of b, save in b(B2)
                    M
0247 C21F        01319          GOTO    getdecnext
0248             01320 fixsign
0248 290A        01321          CLRF    WREG, F
0249 32A6        01322          CPFSGT  DECSIGN
024A 0002        01323          RETURN
                 01324          NEG24   VALBUF
                    M
024B 1331           M          COMF    VALBUF+B0, F
024C 1332           M          COMF    VALBUF+B1, F
024D 1333           M          COMF    VALBUF+B2, F
024E 290A           M          CLRF    WREG, F
024F 1531           M          INCF    VALBUF+B0, F
0250 1132           M          ADDWFC  VALBUF+B1, F
0251 1133           M          ADDWFC  VALBUF+B2, F
                    M
0252 0002        01325          RETURN
                 01326
                 01327          #endif
                 01328
                 01329 ;*************************************************************************
                 01330
                 01331 ;*************************************************************************
                 01332 ; NAME:          GetPar
                 01333 ;
                 01334 ; DESCRIPTION:   Get a parameter number [0,FF] from the serial port and place
                 01335 ;                it in VALBUF+B0.
                 01336 ;
                 01337
0253             01338 GetPar
                 01339
```

© 1997 Microchip Technology Inc.

```
                    01340         CLR24   VALBUF
0253 2931               M         CLRF    VALBUF+B0, F
0254 2932               M         CLRF    VALBUF+B1, F
0255 2933               M         CLRF    VALBUF+B2, F
                        M
                    01341
0256 E1DD           01342         CALL    GetHex
0257 6A51           01343         MOVFP   HEXVAL,WREG
0258 B50F           01344         ANDLW   0x0F
0259 4A31           01345         MOVPF   WREG,VALBUF+B0
025A 1D31           01346         SWAPF   VALBUF+B0, F
                    01347
025B E1DD           01348         CALL    GetHex
025C 6A31           01349         MOVFP   VALBUF+B0,WREG
                    01350
025D 0E51           01351         ADDWF   HEXVAL,W
025E 4A31           01352         MOVPF   WREG,VALBUF+B0
                    01353
025F 0002           01354         RETURN
                    01355
                    01356 ;*****************************************************************************
                    01357
                    01358 ;*****************************************************************************
                    01359 ; NAME:         PutVal
                    01360 ;
                    01361 ; DESCRIPTION:  Sends the value in VALBUF [800000,7FFFFF] out the serial port.
                    01362 ;
                    01363
                    01364         #if     DECIO
                    01365         #else
                    01366
                    01367 PutVal
                    01368
                    01369         MOVFP   VALBUF+B2,WREG
                    01370         CALL    PutHex
                    01371         MOVFP   VALBUF+B1,WREG
                    01372         CALL    PutHex
                    01373         MOVFP   VALBUF+B0,WREG
                    01374         CALL    PutHex
                    01375
                    01376         RETURN
                    01377
                    01378         #endif
                    01379
                    01380 ;*****************************************************************************
                    01381
                    01382 ;*****************************************************************************
```


© 1997 Microchip Technology Inc.

```
                    01383 ; NAME:          PutDecVal
                    01384 ;
                    01385 ; DESCRIPTION:   Send the value in VALBUF [-8388608,8388607] out the serial port.
                    01386 ;
                    01387
                    01388          #if     DECIO
                    01389
0260                01390 PutDecVal
                    01391
0260 9733           01392          BTFSS   VALBUF+B2,MSB
0261 C26C           01393          GOTO    pdpos
                    01394          NEG24   VALBUF
                        M
0262 1331               M          COMF    VALBUF+B0, F
0263 1332               M          COMF    VALBUF+B1, F
0264 1333               M          COMF    VALBUF+B2, F
0265 290A               M          CLRF    WREG, F
0266 1531               M          INCF    VALBUF+B0, F
0267 1132               M          ADDWFC  VALBUF+B1, F
0268 1133               M          ADDWFC  VALBUF+B2, F
                        M
0269 B02D           01395          MOVLW   MN
026A E1A4           01396          CALL    PutChar
026B C26E           01397          GOTO    pddigits
026C                01398 pdpos
026C B020           01399          MOVLW   SP
026D E1A4           01400          CALL    PutChar
                    01401
026E                01402 pddigits
026E B09A           01403          MOVLW   LOW    DEC_TABLE          ; DEC_TABLE LSB
026F 4A0D           01404          MOVPF   WREG,TBLPTRL
0270 B007           01405          MOVLW   HIGH   DEC_TABLE          ; DEC_TABLE MSB
0271 4A0E           01406          MOVPF   WREG,TBLPTRH
                    01407
0272 A937           01408          TABLRD  0,1,DVALBUF+B0
0273                01409 readNextDec
0273 A037           01410          TLRD    0,DVALBUF+B0              ; read entry from table
0274 AB38           01411          TABLRD  1,1,DVALBUF+B1
0275 A939           01412          TABLRD  0,1,DVALBUF+B2
                    01413
0276 2B0A           01414          SETF    WREG, F                   ; unitsposition if end of table
0277 3137           01415          CPFSEQ  DVALBUF+B0
0278 C27A           01416          GOTO    getdigit
0279 C28E           01417          GOTO    unitsposition
027A                01418 getdigit
027A 1537           01419          INCF    DVALBUF+B0, F             ; restore to power of 10
027B 2BA3           01420          SETF    DECVAL, F                 ; set DECVAL to -1
```

© 1997 Microchip Technology Inc.

```
027C                    01421 inc
027C 15A3               01422         INCF    DECVAL, F               ; increment DECVAL
                        01423         SUB24   DVALBUF,VALBUF          ; check if in range
                            M
027D 6A37                   M         MOVFP   DVALBUF+B0,WREG         ; get lowest byte of a into w
027E 0531                   M         SUBWF   VALBUF+B0, F            ; sub lowest byte of b, save in b(B0)
027F 6A38                   M         MOVFP   DVALBUF+B1,WREG         ; get 2nd byte of a into w
0280 0332                   M         SUBWFB  VALBUF+B1, F            ; sub 2nd byte of b, save in b(B1)
0281 6A39                   M         MOVFP   DVALBUF+B2,WREG         ; get 3rd byte of a into w
0282 0333                   M         SUBWFB  VALBUF+B2, F            ; sub 3rd byte of b, save in b(B2)
                            M
0283 9733               01424         BTFSS   VALBUF+B2,MSB
0284 C27C               01425         GOTO    inc
                        01426
                        01427         ADD24   DVALBUF,VALBUF          ; if so, correct VALBUF for next digit
                            M
0285 6A37                   M         MOVFP   DVALBUF+B0,WREG         ; get lowest byte of a into w
0286 0F31                   M         ADDWF   VALBUF+B0, F            ; add lowest byte of b, save in b(B0)
0287 6A38                   M         MOVFP   DVALBUF+B1,WREG         ; get 2nd byte of a into w
0288 1132                   M         ADDWFC  VALBUF+B1, F            ; add 2nd byte of b, save in b(B1)
0289 6A39                   M         MOVFP   DVALBUF+B2,WREG         ; get 3rd byte of a into w
028A 1133                   M         ADDWFC  VALBUF+B2, F            ; add 3rd byte of b, save in b(B2)
                            M
028B 6AA3               01428         MOVFP   DECVAL,WREG             ; send DECVAL
028C E1B0               01429         CALL    PutDec
                        01430
028D C273               01431         GOTO    readNextDec             ; get next table entry
                        01432
028E                    01433 unitsposition
028E 6A31               01434         MOVFP   VALBUF+B0,WREG          ; unit position value now in VALBUF
028F E1B0               01435         CALL    PutDec
                        01436
0290 0002               01437         RETURN
                        01438
                        01439
                        01440         #endif
                        01441
                        01442 ;*****************************************************************************
                        01443
                        01444 ;*****************************************************************************
                        01445 ; NAME:         doError
                        01446 ;
                        01447 ; DESCRIPTION:  Calculates the position and velocity error.
                        01448 ;
                        01449
0291                    01450 doError
                        01451
```


© 1997 Microchip Technology Inc.

```
                    01452          MOV24   POSITION,POSERROR       ; calculate position error
                        M
0291 6A58               M          MOVFP   POSITION+B0,WREG        ; get byte of a into w
0292 4A7C               M          MOVPF   WREG,POSERROR+B0        ; move to b(B0)
0293 6A59               M          MOVFP   POSITION+B1,WREG        ; get byte of a into w
0294 4A7D               M          MOVPF   WREG,POSERROR+B1        ; move to b(B1)
0295 6A5A               M          MOVFP   POSITION+B2,WREG        ; get byte of a into w
0296 4A7E               M          MOVPF   WREG,POSERROR+B2        ; move to b(B2)
                        M
                    01453          SUB24   MPOSITION,POSERROR
                        M
0297 6A75               M          MOVFP   MPOSITION+B0,WREG       ; get lowest byte of a into w
0298 057C               M          SUBWF   POSERROR+B0, F          ; sub lowest byte of b, save in b(B0)
0299 6A76               M          MOVFP   MPOSITION+B1,WREG       ; get 2nd byte of a into w
029A 037D               M          SUBWFB  POSERROR+B1, F          ; sub 2nd byte of b, save in b(B1)
029B 6A77               M          MOVFP   MPOSITION+B2,WREG       ; get 3rd byte of a into w
029C 037E               M          SUBWFB  POSERROR+B2, F          ; sub 3rd byte of b, save in b(B2)
                        M
                    01454
029D 9F7E           01455          BTFSC   POSERROR+B2,MSB         ; saturate error to lowest 16 bits
029E C2AA           01456          GOTO    pneg
029F                01457 ppos
029F 6A7D           01458          MOVFP   POSERROR+B1,WREG
02A0 B580           01459          ANDLW   0x80
02A1 097E           01460          IORWF   POSERROR+B2, F
02A2 290A           01461          CLRF    WREG, F
02A3 327E           01462          CPFSGT  POSERROR+B2
02A4 C2B4           01463          GOTO    psatok
02A5 297E           01464          CLRF    POSERROR+B2, F          ; clear high byte for debug purposes
02A6 B07F           01465          MOVLW   0x7F
02A7 4A7D           01466          MOVPF   WREG,POSERROR+B1
02A8 2B7C           01467          SETF    POSERROR, F
02A9 C2B4           01468          GOTO    psatok
02AA                01469 pneg
02AA 6A7D           01470          MOVFP   POSERROR+B1,WREG
02AB B37F           01471          IORLW   0x7F
02AC 0B7E           01472          ANDWF   POSERROR+B2, F
02AD 2B0A           01473          SETF    WREG, F
02AE 307E           01474          CPFSLT  POSERROR+B2
02AF C2B4           01475          GOTO    psatok
02B0 2B7E           01476          SETF    POSERROR+B2, F          ; set high byte to 0xFF for debug purposes
02B1 297D           01477          CLRF    POSERROR+B1, F
02B2 877D           01478          BSF     POSERROR+B1,MSB
02B3 297C           01479          CLRF    POSERROR, F
02B4                01480 psatok
                    01481
                    01482          MOV24   VELOCITY,VELERROR       ; calculate velocity error
```

© 1997 Microchip Technology Inc.

```
                    M
02B4 6A5B           M         MOVFP   VELOCITY+B0,WREG        ; get byte of a into w
02B5 4A7F           M         MOVPF   WREG,VELERROR+B0        ; move to b(B0)
02B6 6A5C           M         MOVFP   VELOCITY+B1,WREG        ; get byte of a into w
02B7 4A80           M         MOVPF   WREG,VELERROR+B1        ; move to b(B1)
02B8 6A5D           M         MOVFP   VELOCITY+B2,WREG        ; get byte of a into w
02B9 4A81           M         MOVPF   WREG,VELERROR+B2        ; move to b(B2)
                    M
                01483         SUB24   MVELOCITY,VELERROR
                    M
02BA 6A78           M         MOVFP   MVELOCITY+B0,WREG       ; get lowest byte of a into w
02BB 057F           M         SUBWF   VELERROR+B0, F          ; sub lowest byte of b, save in b(B0)
02BC 6A79           M         MOVFP   MVELOCITY+B1,WREG       ; get 2nd byte of a into w
02BD 0380           M         SUBWFB  VELERROR+B1, F          ; sub 2nd byte of b, save in b(B1)
02BE 6A7A           M         MOVFP   MVELOCITY+B2,WREG       ; get 3rd byte of a into w
02BF 0381           M         SUBWFB  VELERROR+B2, F          ; sub 3rd byte of b, save in b(B2)
                    M
                01484
02C0 9F81       01485         BTFSC   VELERROR+B2,MSB         ; saturate error to lowest 16 bits
02C1 C2CD       01486         GOTO    vneg
02C2            01487 vpos
02C2 6A80       01488         MOVFP   VELERROR+B1,WREG
02C3 B580       01489         ANDLW   0x80
02C4 0981       01490         IORWF   VELERROR+B2, F
02C5 290A       01491         CLRF    WREG, F
02C6 3281       01492         CPFSGT  VELERROR+B2
02C7 C2D7       01493         GOTO    vsatok
02C8 2981       01494         CLRF    VELERROR+B2, F
02C9 B07F       01495         MOVLW   0x7F
02CA 4A80       01496         MOVPF   WREG,VELERROR+B1
02CB 2B7F       01497         SETF    VELERROR, F
02CC C2D7       01498         GOTO    vsatok
02CD            01499 vneg
02CD 6A80       01500         MOVFP   VELERROR+B1,WREG
02CE B37F       01501         IORLW   0x7F
02CF 0B81       01502         ANDWF   VELERROR+B2, F
02D0 2B0A       01503         SETF    WREG, F
02D1 3081       01504         CPFSLT  VELERROR+B2
02D2 C2D7       01505         GOTO    vsatok
02D3 2B81       01506         SETF    VELERROR+B2, F
02D4 2980       01507         CLRF    VELERROR+B1, F
02D5 8780       01508         BSF     VELERROR+B1,MSB
02D6 297F       01509         CLRF    VELERROR, F
02D7            01510 vsatok
02D7 0002       01511         RETURN
                01512
                01513 ;*****************************************************************************
```


© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.


```
                    01514
                    01515 ;*****************************************************************************
                    01516 ; NAME:         doServo
                    01517 ;
                    01518 ; DESCRIPTION:  Performs the servo loop calculations.
                    01519 ;
02D8                01520 doServo
                    01521
                    01522         MOV16   POSERROR,U0             ; save new position error in U0
                        M
02D8 6A7C               M         MOVFP   POSERROR+B0,WREG        ; get byte of a into w
02D9 4A8F               M         MOVPF   WREG,U0+B0              ; move to b(B0)
02DA 6A7D               M         MOVFP   POSERROR+B1,WREG        ; get byte of a into w
02DB 4A90               M         MOVPF   WREG,U0+B1              ; move to b(B1)
                        M
                    01523
                    01524         LOADAB  U0,KP                   ; compute KP*U0
                        M
02DC 7C8F               M         MOVFP   U0+B0,AARG+B0           ; load lo byte of A to AARG
02DD 7D90               M         MOVFP   U0+B1,AARG+B1           ; load hi byte of A to AARG
02DE 7E26               M         MOVFP   KP+B0,BARG+B0           ; load lo byte of B to BARG
02DF 7F27               M         MOVFP   KP+B1,BARG+B1           ; load hi byte of B to BARG
                        M
02E0 E630           01525         CALL    Dmult
                    01526         MVPF32  DPX,Y                   ; Y=KP*U0
                        M
02E1 5883               M         MOVPF   DPX+B0,Y+B0             ; move A(B0) to B(B0)
02E2 5984               M         MOVPF   DPX+B1,Y+B1             ; move A(B1) to B(B1)
02E3 5A85               M         MOVPF   DPX+B2,Y+B2             ; move A(B2) to B(B2)
02E4 5B86               M         MOVPF   DPX+B3,Y+B3             ; move A(B3) to B(B3)
                        M
                    01527
02E5 290A           01528         CLRF    WREG, F                 ; if previous output saturated, do
02E6 329A           01529         CPFSGT  SATFLAG                 ; not accumulate integrator
02E7 E618           01530         CALL    doIntegral
                    01531
                    01532         LOADAB  INTEGRAL,KI             ; compute KI*INTEGRAL
                        M
02E8 7C9B               M         MOVFP   INTEGRAL+B0,AARG+B0     ; load lo byte of A to AARG
02E9 7D9C               M         MOVFP   INTEGRAL+B1,AARG+B1     ; load hi byte of A to AARG
02EA 7E2A               M         MOVFP   KI+B0,BARG+B0           ; load lo byte of B to BARG
02EB 7F2B               M         MOVFP   KI+B1,BARG+B1           ; load hi byte of B to BARG
                        M
02EC E630           01533         CALL    Dmult
                    01534         ADD32   DPX,Y                   ; Y=KP*U0+KI*INTEGRAL
                        M
02ED 6A18               M         MOVFP   DPX+B0,WREG             ; get lowest byte of a into w
```

```
02EE 0F83              M        ADDWF   Y+B0, F              ; add lowest byte of b, save in b(B0)
02EF 6A19              M        MOVFP   DPX+B1,WREG          ; get 2nd byte of a into w
02F0 1184              M        ADDWFC  Y+B1, F              ; add 2nd byte of b, save in b(B1)
02F1 6A1A              M        MOVFP   DPX+B2,WREG          ; get 3rd byte of a into w
02F2 1185              M        ADDWFC  Y+B2, F              ; add 3rd byte of b, save in b(B2)
02F3 6A1B              M        MOVFP   DPX+B3,WREG          ; get 4th byte of a into w
02F4 1186              M        ADDWFC  Y+B3, F              ; add 4th byte of b, save in b(B3)
                       M
                    01535
                    01536        MVFP16  U0,AARG              ; compute KV*(U0-U1)
                       M
02F5 7C8F              M        MOVFP   U0+B0,AARG+B0        ; move A(B0) to B(B0)
02F6 7D90              M        MOVFP   U0+B1,AARG+B1        ; move A(B1) to B(B1)
                       M
                    01537        SUB16   U1,AARG
                       M
02F7 6A91              M        MOVFP   U1+B0,WREG           ; get lowest byte of a into w
02F8 051C              M        SUBWF   AARG+B0, F           ; sub lowest byte of b, save in b(B0)
02F9 6A92              M        MOVFP   U1+B1,WREG           ; get 2nd byte of a into w
02FA 031D              M        SUBWFB  AARG+B1, F           ; sub 2nd byte of b, save in b(B1)
                       M
                    01538        MVFP16  KV,BARG
                       M
02FB 7E28              M        MOVFP   KV+B0,BARG+B0        ; move A(B0) to B(B0)
02FC 7F29              M        MOVFP   KV+B1,BARG+B1        ; move A(B1) to B(B1)
                       M
02FD E630           01539       CALL    Dmult
                    01540       ADD32   DPX,Y                ; Y=KP*U0+KI*INTEGRAL+KV*(U0-U1)
                       M
02FE 6A18              M        MOVFP   DPX+B0,WREG          ; get lowest byte of a into w
02FF 0F83              M        ADDWF   Y+B0, F              ; add lowest byte of b, save in b(B0)
0300 6A19              M        MOVFP   DPX+B1,WREG          ; get 2nd byte of a into w
0301 1184              M        ADDWFC  Y+B1, F              ; add 2nd byte of b, save in b(B1)
0302 6A1A              M        MOVFP   DPX+B2,WREG          ; get 3rd byte of a into w
0303 1185              M        ADDWFC  Y+B2, F              ; add 3rd byte of b, save in b(B2)
0304 6A1B              M        MOVFP   DPX+B3,WREG          ; get 4th byte of a into w
0305 1186              M        ADDWFC  Y+B3, F              ; add 4th byte of b, save in b(B3)
                       M
                    01541
0306 290A           01542       CLRF    WREG, F
0307 32C4           01543       CPFSGT  SHIFTNUM
0308 C311           01544       GOTO    grabok
0309 78C4           01545       MOVFP   SHIFTNUM,TMP
030A                01546 grabloop
                    01547       RLC32   Y
                       M
030A 8804              M        BCF     _carry
```


© 1997 Microchip Technology Inc.

```
030B 1B83              M         RLCF    Y+B0, F
030C 1B84              M         RLCF    Y+B1, F
030D 1B85              M         RLCF    Y+B2, F
030E 1B86              M         RLCF    Y+B3, F
                       M
030F 1718        01548           DECFSZ  TMP, F
0310 C30A        01549           GOTO    grabloop
                 01550
0311             01551 grabok
0311 299A        01552           CLRF    SATFLAG, F
0312 9F86        01553           BTFSC   Y+B3,MSB                ; saturate to middle 16 bits,
0313 C321        01554           GOTO    negs                    ; keeping top 10 bits for PW1DCH
0314             01555 poss                                      ; and PW1DCL
0314 6A85        01556           MOVFP   Y+B2,WREG               ; check if Y >= 2**23
0315 B580        01557           ANDLW   0x80
0316 0986        01558           IORWF   Y+B3, F
0317 290A        01559           CLRF    WREG, F
0318 3286        01560           CPFSGT  Y+B3
0319 C32D        01561           GOTO    zero6bits               ; if not, zero 6 bits
                 01562
031A 159A        01563           INCF    SATFLAG, F              ; if so, set Y=0x007FFFFF
031B 2986        01564           CLRF    Y+B3, F                 ; clear for debug purposes
031C B07F        01565           MOVLW   0x7F
031D 4A85        01566           MOVPF   WREG,Y+B2
031E 2B84        01567           SETF    Y+B1, F
031F 2B83        01568           SETF    Y+B0, F
0320 C32D        01569           GOTO    zero6bits
0321             01570 negs
0321 6A85        01571           MOVFP   Y+B2,WREG               ; check if Y <= -2**23
0322 B37F        01572           IORLW   0x7F
0323 0B86        01573           ANDWF   Y+B3, F
0324 2B0A        01574           SETF    WREG, F
0325 3086        01575           CPFSLT  Y+B3
0326 C32D        01576           GOTO    zero6bits               ; if not, zero 6 bits
                 01577
0327 2B9A        01578           SETF    SATFLAG, F              ; if so, set Y = 0xFF800000
0328 2B86        01579           SETF    Y+B3, F
0329 2985        01580           CLRF    Y+B2, F
032A 8785        01581           BSF     Y+B2,MSB
032B 2984        01582           CLRF    Y+B1, F
032C 2983        01583           CLRF    Y+B0, F
                 01584
032D             01585 zero6bits
                 01586           MOV24   Y+B1,YPWM+B0            ; move Y to YPWM and zero 6 bits
                    M
032D 6A84           M            MOVFP   Y+B1+B0,WREG            ; get byte of a into w
032E 4A87           M            MOVPF   WREG,YPWM+B0+B0         ; move to b(B0)
```

© 1997 Microchip Technology Inc.

```
032F 6A85              M         MOVFP   Y+B1+B1,WREG          ; get byte of a into w
0330 4A88              M         MOVPF   WREG,YPWM+B0+B1       ; move to b(B1)
0331 6A86              M         MOVFP   Y+B1+B2,WREG          ; get byte of a into w
0332 4A89              M         MOVPF   WREG,YPWM+B0+B2       ; move to b(B2)
                       M
0333                01587 doTorque                             ; entry point for torque mode
0333 B0C0           01588         MOVLW   0xC0
0334 0B87           01589         ANDWF   YPWM+B0, F
                    01590
0335 9F88           01591         BTFSC   YPWM+B1,MSB
0336 C33E           01592         GOTO    tmlimit
0337                01593 tplimit
0337 9697           01594         BTFSS   EXTSTAT,BIT6
0338 C344           01595         GOTO    mplimitok
                    01596         CLR32   YPWM
0339 2987              M          CLRF    YPWM+B0, F
033A 2988              M          CLRF    YPWM+B1, F
033B 2989              M          CLRF    YPWM+B2, F
033C 298A              M          CLRF    YPWM+B3, F
                       M
033D C344           01597         GOTO    mplimitok
033E                01598 tmlimit
033E 9597           01599         BTFSS   EXTSTAT,BIT5
033F C344           01600         GOTO    mplimitok
                    01601         CLR32   YPWM
0340 2987              M          CLRF    YPWM+B0, F
0341 2988              M          CLRF    YPWM+B1, F
0342 2989              M          CLRF    YPWM+B2, F
0343 298A              M          CLRF    YPWM+B3, F
                       M
0344                01602 mplimitok
0344 B07F           01603         MOVLW   PW1DCH_INIT           ; adjustment from bipolar to unipolar
0345 4A19           01604         MOVPF   WREG,TMP+B1           ; for 50% duty cycle
0346 B0C0           01605         MOVLW   PW1DCL_INIT
0347 4A18           01606         MOVPF   WREG,TMP+B0
                    01607         ADD16   TMP,YPWM
                       M
0348 6A18              M          MOVFP   TMP+B0,WREG           ; get lowest byte of a into w
0349 0F87              M          ADDWF   YPWM+B0, F            ; add lowest byte of b, save in b(B0)
034A 6A19              M          MOVFP   TMP+B1,WREG           ; get 2nd byte of a into w
034B 1188              M          ADDWFC  YPWM+B1, F            ; add 2nd byte of b, save in b(B1)
                       M
                    01608
034C 2919           01609         CLRF    TMP+B1, F             ; correct by 1 LSB
034D B040           01610         MOVLW   0x40                  ; add one to bit5 of PW1DCL
034E 4A18           01611         MOVPF   WREG,TMP+B0
                    01612         ADD16   TMP,YPWM
```


© 1997 Microchip Technology Inc.

```
                        M
034F 6A18               M           MOVFP   TMP+B0,WREG             ; get lowest byte of a into w
0350 0F87               M           ADDWF   YPWM+B0, F              ; add lowest byte of b, save in b(B0)
0351 6A19               M           MOVFP   TMP+B1,WREG             ; get 2nd byte of a into w
0352 1188               M           ADDWFC  YPWM+B1, F              ; add 2nd byte of b, save in b(B1)
                        M
                    01613
0353                01614 testmax
0353 291A           01615           CLRF    TMP+B2, F               ; check pwm maximum limit
0354 2989           01616           CLRF    YPWM+B2, F              ; LMD18200 must have a minimum pulse
0355 298A           01617           CLRF    YPWM+B3, F              ; so duty cycle must not be 0 or 100%
                    01618           MVFP16  YPWMAX,TMP
                        M
0356 788D               M           MOVFP   YPWMAX+B0,TMP+B0        ; move A(B0) to B(B0)
0357 798E               M           MOVFP   YPWMAX+B1,TMP+B1        ; move A(B1) to B(B1)
                        M
                    01619           SUB24   YPWM,TMP
                        M
0358 6A87               M           MOVFP   YPWM+B0,WREG            ; get lowest byte of a into w
0359 0518               M           SUBWF   TMP+B0, F               ; sub lowest byte of b, save in b(B0)
035A 6A88               M           MOVFP   YPWM+B1,WREG            ; get 2nd byte of a into w
035B 0319               M           SUBWFB  TMP+B1, F               ; sub 2nd byte of b, save in b(B1)
035C 6A89               M           MOVFP   YPWM+B2,WREG            ; get 3rd byte of a into w
035D 031A               M           SUBWFB  TMP+B2, F               ; sub 3rd byte of b, save in b(B2)
                        M
035E 971A           01620           BTFSS   TMP+B2,MSB
035F C365           01621           GOTO    testmin
                    01622           MOV16   YPWMAX,YPWM             ; saturate to max
                        M
0360 6A8D               M           MOVFP   YPWMAX+B0,WREG          ; get byte of a into w
0361 4A87               M           MOVPF   WREG,YPWM+B0            ; move to b(B0)
0362 6A8E               M           MOVFP   YPWMAX+B1,WREG          ; get byte of a into w
0363 4A88               M           MOVPF   WREG,YPWM+B1            ; move to b(B1)
                        M
0364 C376           01623           GOTO    limitok
0365                01624 testmin
0365 291A           01625           CLRF    TMP+B2, F               ; check pwm minimum limit
0366 2989           01626           CLRF    YPWM+B2, F
0367 298A           01627           CLRF    YPWM+B3, F
                    01628           MVFP16  YPWMIN,TMP
                        M
0368 788B               M           MOVFP   YPWMIN+B0,TMP+B0        ; move A(B0) to B(B0)
0369 798C               M           MOVFP   YPWMIN+B1,TMP+B1        ; move A(B1) to B(B1)
                        M
                    01629           SUB24   YPWM,TMP
                        M
036A 6A87               M           MOVFP   YPWM+B0,WREG            ; get lowest byte of a into w
```

© 1997 Microchip Technology Inc.

```
036B 0518              M         SUBWF   TMP+B0, F               ; sub lowest byte of b, save in b(B0)
036C 6A88              M         MOVFP   YPWM+B1,WREG            ; get 2nd byte of a into w
036D 0319              M         SUBWFB  TMP+B1, F               ; sub 2nd byte of b, save in b(B1)
036E 6A89              M         MOVFP   YPWM+B2,WREG            ; get 3rd byte of a into w
036F 031A              M         SUBWFB  TMP+B2, F               ; sub 3rd byte of b, save in b(B2)
                       M
0370 9F1A          01630         BTFSC   TMP+B2,MSB
0371 C376          01631         GOTO    limitok
                   01632         MOV16   YPWMIN,YPWM             ; saturate to min
                       M
0372 6A8B              M         MOVFP   YPWMIN+B0,WREG          ; get byte of a into w
0373 4A87              M         MOVPF   WREG,YPWM+B0            ; move to b(B0)
0374 6A8C              M         MOVFP   YPWMIN+B1,WREG          ; get byte of a into w
0375 4A88              M         MOVPF   WREG,YPWM+B1            ; move to b(B1)
                       M
0376               01633 limitok
0376 B803          01634         MOVLB   BANK3                   ; set new duty cycle
0377 7087          01635         MOVFP   YPWM+B0,PW1DCL
0378 7288          01636         MOVFP   YPWM+B1,PW1DCH
                   01637
                   01638         MOV16   U0,U1                   ; push errors into U(k-1)
                       M
0379 6A8F              M         MOVFP   U0+B0,WREG              ; get byte of a into w
037A 4A91              M         MOVPF   WREG,U1+B0              ; move to b(B0)
037B 6A90              M         MOVFP   U0+B1,WREG              ; get byte of a into w
037C 4A92              M         MOVPF   WREG,U1+B1              ; move to b(B1)
                       M
                   01639
037D 0002          01640         RETURN
                   01641
                   01642 ;*************************************************************************
                   01643
                   01644 ;*************************************************************************
                   01645 ; NAME:          doPreMove
                   01646 ;
                   01647 ; DESCRIPTION:
                   01648
037E               01649 doPreMove
                   01650
                   01651         MOV24   NMOVVAL,MOVVAL          ; move buffer to MOVVAL
                       M
037E 6A5E              M         MOVFP   NMOVVAL+B0,WREG         ; get byte of a into w
037F 4A62              M         MOVPF   WREG,MOVVAL+B0          ; move to b(B0)
0380 6A5F              M         MOVFP   NMOVVAL+B1,WREG         ; get byte of a into w
0381 4A63              M         MOVPF   WREG,MOVVAL+B1          ; move to b(B1)
0382 6A60              M         MOVFP   NMOVVAL+B2,WREG         ; get byte of a into w
0383 4A64              M         MOVPF   WREG,MOVVAL+B2          ; move to b(B2)
```


© 1997 Microchip Technology Inc.

```
                        M
0384 8F98       01652   BCF     MOVSTAT,BIT7            ; clear buffer flag
0385 8698       01653   BSF     MOVSTAT,BIT6            ; set motion status flag
0386 8598       01654   BSF     MOVSTAT,BIT5            ; set move in progress flag
0387 6ACB       01655   MOVFP   ONE,WREG
0388 4A99       01656   MOVPF   WREG,MOVFLAG            ; initialize MOVEFLAG to 1
                01657
                01658
0389 2954       01659   CLRF    OPOSITION+B0, F         ; initialize buffers
                01660   MOV24   POSITION,OPOSITION+B1
                    M
038A 6A58           M   MOVFP   POSITION+B0,WREG        ; get byte of a into w
038B 4A55           M   MOVPF   WREG,OPOSITION+B1+B0    ; move to b(B0)
038C 6A59           M   MOVFP   POSITION+B1,WREG        ; get byte of a into w
038D 4A56           M   MOVPF   WREG,OPOSITION+B1+B1    ; move to b(B1)
038E 6A5A           M   MOVFP   POSITION+B2,WREG        ; get byte of a into w
038F 4A57           M   MOVPF   WREG,OPOSITION+B1+B2    ; move to b(B2)
                    M
                01661   MOV32   OPOSITION,MOVPBUF
                    M
0390 6A54           M   MOVFP   OPOSITION+B0,WREG       ; get byte of a into w
0391 4AAF           M   MOVPF   WREG,MOVPBUF+B0         ; move to b(B0)
0392 6A55           M   MOVFP   OPOSITION+B1,WREG       ; get byte of a into w
0393 4AB0           M   MOVPF   WREG,MOVPBUF+B1         ; move to b(B1)
0394 6A56           M   MOVFP   OPOSITION+B2,WREG       ; get byte of a into w
0395 4AB1           M   MOVPF   WREG,MOVPBUF+B2         ; move to b(B2)
0396 6A57           M   MOVFP   OPOSITION+B3,WREG       ; get byte of a into w
0397 4AB2           M   MOVPF   WREG,MOVPBUF+B3         ; move to b(B3)
                    M
0398 299A       01662   CLRF    SATFLAG, F
                01663   CLR16   MOVTIME                 ; clear move times
0399 296A           M   CLRF    MOVTIME+B0, F
039A 296B           M   CLRF    MOVTIME+B1, F
                    M
                01664   CLR16   T1                      ; 0 used as flag for no maximum speed
039B 296D           M   CLRF    T1+B0, F
039C 296E           M   CLRF    T1+B1, F
                    M
                01665   CLR16   T2
039D 296F           M   CLRF    T2+B0, F
039E 2970           M   CLRF    T2+B1, F
                    M
                01666   CLR16   TAU
039F 2971           M   CLRF    TAU+B0, F
03A0 2972           M   CLRF    TAU+B1, F
                    M
                01667   CLR32   MOVDEL                  ; clear move discretization error
```

© 1997 Microchip Technology Inc.

```
03A1 29BB              M         CLRF    MOVDEL+B0, F
03A2 29BC              M         CLRF    MOVDEL+B1, F
03A3 29BD              M         CLRF    MOVDEL+B2, F
03A4 29BE              M         CLRF    MOVDEL+B3, F
                       M
                    01668        CLR16   PH2FLAT                 ; clear phase 2 flat counter
03A5 29BF              M         CLRF    PH2FLAT+B0, F
03A6 29C0              M         CLRF    PH2FLAT+B1, F
                       M
                    01669
03A7 3396           01670        TSTFSZ  MODETYPE
03A8 C404           01671        GOTO    vmode
03A9                01672 pmode
                    01673        MVFP24  MOVVAL,TMP
                       M
03A9 7862              M         MOVFP   MOVVAL+B0,TMP+B0        ; move A(B0) to B(B0)
03AA 7963              M         MOVFP   MOVVAL+B1,TMP+B1        ; move A(B1) to B(B1)
03AB 7A64              M         MOVFP   MOVVAL+B2,TMP+B2        ; move A(B2) to B(B2)
                       M
03AC 971A           01674        BTFSS   TMP+B2,MSB
03AD C3B5           01675        GOTO    mvpos
                    01676        NEG24   TMP
                       M
03AE 1318              M         COMF    TMP+B0, F
03AF 1319              M         COMF    TMP+B1, F
03B0 131A              M         COMF    TMP+B2, F
03B1 290A              M         CLRF    WREG, F
03B2 1518              M         INCF    TMP+B0, F
03B3 1119              M         ADDWFC  TMP+B1, F
03B4 111A              M         ADDWFC  TMP+B2, F
                       M
03B5                01677 mvpos
03B5 291C           01678        CLRF    MOVTMP+B0, F            ; calculate  abs(MOVVAL) - 3
03B6 291D           01679        CLRF    MOVTMP+B1, F            ; do immediate move if negative
03B7 291E           01680        CLRF    MOVTMP+B2, F
03B8 801C           01681        BSF     MOVTMP+B0,BIT0
03B9 811C           01682        BSF     MOVTMP+B0,BIT1
                    01683        SUB24   MOVTMP,TMP
                       M
03BA 6A1C              M         MOVFP   MOVTMP+B0,WREG          ; get lowest byte of a into w
03BB 0518              M         SUBWF   TMP+B0, F               ; sub lowest byte of b, save in b(B0)
03BC 6A1D              M         MOVFP   MOVTMP+B1,WREG          ; get 2nd byte of a into w
03BD 0319              M         SUBWFB  TMP+B1, F               ; sub 2nd byte of b, save in b(B1)
03BE 6A1E              M         MOVFP   MOVTMP+B2,WREG          ; get 3rd byte of a into w
03BF 031A              M         SUBWFB  TMP+B2, F               ; sub 3rd byte of b, save in b(B2)
                       M
                    01684
```


© 1997 Microchip Technology Inc.

```
03C0 971A          01685          BTFSS   TMP+B2,MSB              ; check for zero move
03C1 C3CD          01686          GOTO    nonzero
03C2 2B95          01687          SETF    SERVOFLAG, F            ; set servoflag to restore servo
03C3 2999          01688          CLRF    MOVFLAG, F
03C4 8D98          01689          BCF     MOVSTAT,BIT5
03C5 8E98          01690          BCF     MOVSTAT,BIT6
                   01691          ADD24   MOVVAL,POSITION
                       M
03C6 6A62              M          MOVFP   MOVVAL+B0,WREG          ; get lowest byte of a into w
03C7 0F58              M          ADDWF   POSITION+B0, F          ; add lowest byte of b, save in b(B0)
03C8 6A63              M          MOVFP   MOVVAL+B1,WREG          ; get 2nd byte of a into w
03C9 1159              M          ADDWFC  POSITION+B1, F          ; add 2nd byte of b, save in b(B1)
03CA 6A64              M          MOVFP   MOVVAL+B2,WREG          ; get 3rd byte of a into w
03CB 115A              M          ADDWFC  POSITION+B2, F          ; add 3rd byte of b, save in b(B2)
                       M
03CC 0002          01692          RETURN
03CD               01693 nonzero
                   01694          CLR32   MOVVBUF
03CD 29B3              M          CLRF    MOVVBUF+B0, F
03CE 29B4              M          CLRF    MOVVBUF+B1, F
03CF 29B5              M          CLRF    MOVVBUF+B2, F
03D0 29B6              M          CLRF    MOVVBUF+B3, F
                       M
                   01695
03D1 6A64          01696          MOVFP   MOVVAL+B2,WREG          ; move sign (00h=positive,80h=negative)
03D2 B580          01697          ANDLW   0x80
03D3 4A6C          01698          MOVPF   WREG,MOVSIGN
                   01699
03D4 29AE          01700          CLRF    V+B3, F                 ; create appropriate velocity and
                   01701          MOV24   VL,V                    ; acceleration limits from move sign
                       M
03D5 6A20              M          MOVFP   VL+B0,WREG              ; get byte of a into w
03D6 4AAB              M          MOVPF   WREG,V+B0               ; move to b(B0)
03D7 6A21              M          MOVFP   VL+B1,WREG              ; get byte of a into w
03D8 4AAC              M          MOVPF   WREG,V+B1               ; move to b(B1)
03D9 6A22              M          MOVFP   VL+B2,WREG              ; get byte of a into w
03DA 4AAD              M          MOVPF   WREG,V+B2               ; move to b(B2)
                       M
03DB 29AA          01702          CLRF    A+B3, F
                   01703          MOV24   AL,A
                       M
03DC 6A23              M          MOVFP   AL+B0,WREG              ; get byte of a into w
03DD 4AA7              M          MOVPF   WREG,A+B0               ; move to b(B0)
03DE 6A24              M          MOVFP   AL+B1,WREG              ; get byte of a into w
03DF 4AA8              M          MOVPF   WREG,A+B1               ; move to b(B1)
03E0 6A25              M          MOVFP   AL+B2,WREG              ; get byte of a into w
03E1 4AA9              M          MOVPF   WREG,A+B2               ; move to b(B2)
```

© 1997 Microchip Technology Inc.

```
                    M
03E2 290A   01704       CLRF    WREG, F
03E3 326C   01705       CPFSGT  MOVSIGN
03E4 C3F7   01706       GOTO    minc
            01707       NEG32   V
                M
03E5 13AB       M       COMF    V+B0, F
03E6 13AC       M       COMF    V+B1, F
03E7 13AD       M       COMF    V+B2, F
03E8 13AE       M       COMF    V+B3, F
03E9 290A       M       CLRF    WREG, F
03EA 15AB       M       INCF    V+B0, F
03EB 11AC       M       ADDWFC  V+B1, F
03EC 11AD       M       ADDWFC  V+B2, F
03ED 11AE       M       ADDWFC  V+B3, F
                M
            01708       NEG32   A
                M
03EE 13A7       M       COMF    A+B0, F
03EF 13A8       M       COMF    A+B1, F
03F0 13A9       M       COMF    A+B2, F
03F1 13AA       M       COMF    A+B3, F
03F2 290A       M       CLRF    WREG, F
03F3 15A7       M       INCF    A+B0, F
03F4 11A8       M       ADDWFC  A+B1, F
03F5 11A9       M       ADDWFC  A+B2, F
03F6 11AA       M       ADDWFC  A+B3, F
                M
03F7        01709 minc
03F7 2966   01710       CLRF    HMOVVAL+B0, F           ; evaluate MOVVAL/2
            01711       MOV24   MOVVAL,HMOVVAL+B1
                M
03F8 6A62       M       MOVFP   MOVVAL+B0,WREG          ; get byte of a into w
03F9 4A67       M       MOVPF   WREG,HMOVVAL+B1+B0      ; move to b(B0)
03FA 6A63       M       MOVFP   MOVVAL+B1,WREG          ; get byte of a into w
03FB 4A68       M       MOVPF   WREG,HMOVVAL+B1+B1      ; move to b(B1)
03FC 6A64       M       MOVFP   MOVVAL+B2,WREG          ; get byte of a into w
03FD 4A69       M       MOVPF   WREG,HMOVVAL+B1+B2      ; move to b(B2)
                M
            01712       RRC32   HMOVVAL                 ; half move in Q8
                M
03FE 1A69       M       RLCF    HMOVVAL+B3,W            ; move sign into carry bit
03FF 1969       M       RRCF    HMOVVAL+B3, F
0400 1968       M       RRCF    HMOVVAL+B2, F
0401 1967       M       RRCF    HMOVVAL+B1, F
0402 1966       M       RRCF    HMOVVAL+B0, F
                M
```


© 1997 Microchip Technology Inc.

```
0403 C43D          01713          GOTO    moderready
                   01714
0404               01715 vmode
0404 9F96          01716          BTFSC   MODETYPE,MSB          ; is it torque move?
0405 C445          01717          GOTO    tmode
                   01718
0406 2969          01719          CLRF    HMOVVAL+B3, F         ; compute final minus initial velocity
                   01720          MOV24   MOVVAL,HMOVVAL
                       M
0407 6A62              M          MOVFP   MOVVAL+B0,WREG        ; get byte of a into w
0408 4A66              M          MOVPF   WREG,HMOVVAL+B0       ; move to b(B0)
0409 6A63              M          MOVFP   MOVVAL+B1,WREG        ; get byte of a into w
040A 4A67              M          MOVPF   WREG,HMOVVAL+B1       ; move to b(B1)
040B 6A64              M          MOVFP   MOVVAL+B2,WREG        ; get byte of a into w
040C 4A68              M          MOVPF   WREG,HMOVVAL+B2       ; move to b(B2)
                       M
040D 9F64          01721          BTFSC   MOVVAL+B2,MSB
040E 2B69          01722          SETF    HMOVVAL+B3, F
                   01723          SUB32   MOVVBUF,HMOVVAL
                       M
040F 6AB3              M          MOVFP   MOVVBUF+B0,WREG       ; get lowest byte of a into w
0410 0566              M          SUBWF   HMOVVAL+B0, F         ; sub lowest byte of b, save in b(B0)
0411 6AB4              M          MOVFP   MOVVBUF+B1,WREG       ; get 2nd byte of a into w
0412 0367              M          SUBWFB  HMOVVAL+B1, F         ; sub 2nd byte of b, save in b(B1)
0413 6AB5              M          MOVFP   MOVVBUF+B2,WREG       ; get 3rd byte of a into w
0414 0368              M          SUBWFB  HMOVVAL+B2, F         ; sub 3rd byte of b, save in b(B2)
0415 6AB6              M          MOVFP   MOVVBUF+B3,WREG       ; get 4th byte of a into w
0416 0369              M          SUBWFB  HMOVVAL+B3, F         ; sub 4th byte of b, save in b(B3)
                       M
                   01724
0417 6A69          01725          MOVFP   HMOVVAL+B3,WREG
0418 B580          01726          ANDLW   0x80
0419 4A6C          01727          MOVPF   WREG,MOVSIGN
                   01728
041A 29AE          01729          CLRF    V+B3, F               ; create appropriate velocity and
                   01730          MOV24   VL,V                  ; acceleration limits from move sign
                       M
041B 6A20              M          MOVFP   VL+B0,WREG            ; get byte of a into w
041C 4AAB              M          MOVPF   WREG,V+B0             ; move to b(B0)
041D 6A21              M          MOVFP   VL+B1,WREG            ; get byte of a into w
041E 4AAC              M          MOVPF   WREG,V+B1             ; move to b(B1)
041F 6A22              M          MOVFP   VL+B2,WREG            ; get byte of a into w
0420 4AAD              M          MOVPF   WREG,V+B2             ; move to b(B2)
                       M
0421 29AA          01731          CLRF    A+B3, F
                   01732          MOV24   AL,A
                       M
```

© 1997 Microchip Technology Inc.

```
0422 6A23              M         MOVFP   AL+B0,WREG              ; get byte of a into w
0423 4AA7              M         MOVPF   WREG,A+B0               ; move to b(B0)
0424 6A24              M         MOVFP   AL+B1,WREG              ; get byte of a into w
0425 4AA8              M         MOVPF   WREG,A+B1               ; move to b(B1)
0426 6A25              M         MOVFP   AL+B2,WREG              ; get byte of a into w
0427 4AA9              M         MOVPF   WREG,A+B2               ; move to b(B2)
                       M
0428 290A          01733         CLRF    WREG, F
0429 326C          01734         CPFSGT  MOVSIGN
042A C43D          01735         GOTO    modeready
                   01736         NEG32   V
                       M
042B 13AB              M         COMF    V+B0, F
042C 13AC              M         COMF    V+B1, F
042D 13AD              M         COMF    V+B2, F
042E 13AE              M         COMF    V+B3, F
042F 290A              M         CLRF    WREG, F
0430 15AB              M         INCF    V+B0, F
0431 11AC              M         ADDWFC  V+B1, F
0432 11AD              M         ADDWFC  V+B2, F
0433 11AE              M         ADDWFC  V+B3, F
                       M
                   01737         NEG32   A
                       M
0434 13A7              M         COMF    A+B0, F
0435 13A8              M         COMF    A+B1, F
0436 13A9              M         COMF    A+B2, F
0437 13AA              M         COMF    A+B3, F
0438 290A              M         CLRF    WREG, F
0439 15A7              M         INCF    A+B0, F
043A 11A8              M         ADDWFC  A+B1, F
043B 11A9              M         ADDWFC  A+B2, F
043C 11AA              M         ADDWFC  A+B3, F
                       M
                   01738
043D               01739 modeready
043D 2965          01740         CLRF    MOVVAL+B3, F
043E 9F64          01741         BTFSC   MOVVAL+B2,MSB
043F 2B65          01742         SETF    MOVVAL+B3, F
                   01743
0440 2B95          01744         SETF    SERVOFLAG, F            ; set servoflag to restore servo
                   01745                                         ; if stopped
                   01746 ;*************************************************************************
                   01747 ;       For PICMASTER Debug/servo tuning puporses only Purposes Only
                   01748 ;
0441               01749 testCapCount
0441 6AC6          01750         MOVFP   CAPCOUNT+B0,WREG
```


© 1997 Microchip Technology Inc.

```
0442 08C7          01751         IORWF   CAPCOUNT+B1,W
0443 4AC5          01752         MOVPF   WREG,CAPFLAG
                   01753 ;*************************************************************************
                   01754
0444 0002          01755         RETURN
                   01756
0445               01757 tmode                                   ; torque/voltage mode
                   01758         MOV16   MOVVAL+B1,YPWM          ; set new commanded value
                       M
0445 6A63              M         MOVFP   MOVVAL+B1+B0,WREG       ; get byte of a into w
0446 4A87              M         MOVPF   WREG,YPWM+B0            ; move to b(B0)
0447 6A64              M         MOVFP   MOVVAL+B1+B1,WREG       ; get byte of a into w
0448 4A88              M         MOVPF   WREG,YPWM+B1            ; move to b(B1)
                       M
0449 2995          01759         CLRF    SERVOFLAG, F            ; disable servo
044A E333          01760         CALL    doTorque                ; set pwm duty cycle
044B 2999          01761         CLRF    MOVFLAG, F
044C 8D98          01762         BCF     MOVSTAT,BIT5
044D C441          01763     goto    testCapCount
                   01764
044E 0002          01765         RETURN
                   01766
                   01767 ;*************************************************************************
                   01768
                   01769 ;*************************************************************************
                   01770 ; NAME:          doMove
                   01771 ;
                   01772 ; DESCRIPTION:   In position mode, trapezoidal moves are performed. Phase1
                   01773 ;                and phase2 respectively, are the periods for the first and
                   01774 ;                second halves of the move. The move time is defined as zero
                   01775 ;                at the beginning of the move,T2 is the time at half the move, T1 is the time w
                   01776 ;                begins,(the region of constant velocity reduces to a point
                   01777 ;                in the case where maximum speed is not realized, and the
                   01778 ;                trapezoidal move degenerates into a trianglular move,
                   01779 ;                together with T1=T2), and TAU is the total time of the move.
                   01780 ;                The accelerations are +-AL or 0.
                   01781 ;
                   01782 ;
                   01783 ;                 triangle speed                 trapezoidal speed
                   01784 ;
                   01785 ;                                                  ____________
                   01786 ;                                                 /            \
                   01787 ;                   /\                           /              \
                   01788 ;                  /  \                         /                \
                   01789 ;                 /    \                       /                  \
                   01790 ;                /      \                     /                    \
                   01791 ;               /        \                   /                      \
```

© 1997 Microchip Technology Inc.

```
01792 ;
01793 ;             0   T1=T2  TAU              0     T1      T2            TAU
01794 ;
01795 ;
01796 ;
01797 ;               Let x dencte the undershoot and y the overshoot commanded
01798 ;               at adjacent sample times as half the move is crossed.
01799 ;               In the case of a triangular move, the discretization error
01800 ;               is given by
01801 ;
01802 ;                        error = min (2x,2y)
01803 ;
01804 ;               For a trapezoidal move, the discretization error is
01805 ;
01806 ;                       error = min (2x,y-x) <= .5*(maximum commanded speed)
01807 ;
01808 ;               This discretization error is resolved in the final sample
01809 ;               time of the move by executing a step to the final position
01810 ;               at zero speed. The method employed here the best possible
01811 ;               performance with regard to discretization error without
01812 ;               dynamically modifying velocity and acceleration limits.
01813 ;
01814 ;
01815 ;
01816 ;               In velocity mode, ramp moves are performed.
01817 ;
01818 ;
01819 ;                                       / final velocity
01820 ;                                      /
01821 ;                                     /
01822 ;                                    /
01823 ;                                   /
01824 ;             initial velocity     /
01825 ;
01826 ;                                  0     TAU
01827 ;
01828 ;
01829
044F            01830 doMove
01831
01832                   INC16   MOVTIME                 ; increment move time
                    M
044F 290A           M           CLRF    WREG, F
0450 156A           M           INCF    MOVTIME+B0, F
0451 116B           M           ADDWFC  MOVTIME+B1, F
                    M
01833
```


© 1997 Microchip Technology Inc.

```
0452 E5A8          01834          CALL    doPosVel              ; evaluate iterative equations
                   01835
0453 3396          01836          TSTFSZ  MODETYPE
0454 C569          01837          GOTO    vmove
0455               01838 pmove
0455 6ACB          01839          MOVFP   ONE,WREG              ; test if in phase1
0456 3199          01840          CPFSEQ  MOVFLAG
0457 C51B          01841          GOTO    phase2
0458               01842 phase1
                   01843          MVFP32  MOVDEL,MOVTMP         ; save previous discretization error
                      M
0458 7CBB             M           MOVFP   MOVDEL+B0,MOVTMP+B0   ; move A(B0) to B(B0)
0459 7DBC             M           MOVFP   MOVDEL+B1,MOVTMP+B1   ; move A(B1) to B(B1)
045A 7EBD             M           MOVFP   MOVDEL+B2,MOVTMP+B2   ; move A(B2) to B(B2)
045B 7FBE             M           MOVFP   MOVDEL+B3,MOVTMP+B3   ; move A(B3) to B(B3)
                      M
                   01844          MOV32   OPOSITION,MOVDEL      ; test if half move
                      M
045C 6A54             M           MOVFP   OPOSITION+B0,WREG     ; get byte of a into w
045D 4ABB             M           MOVPF   WREG,MOVDEL+B0        ; move to b(B0)
045E 6A55             M           MOVFP   OPOSITION+B1,WREG     ; get byte of a into w
045F 4ABC             M           MOVPF   WREG,MOVDEL+B1        ; move to b(B1)
0460 6A56             M           MOVFP   OPOSITION+B2,WREG     ; get byte of a into w
0461 4ABD             M           MOVPF   WREG,MOVDEL+B2        ; move to b(B2)
0462 6A57             M           MOVFP   OPOSITION+B3,WREG     ; get byte of a into w
0463 4ABE             M           MOVPF   WREG,MOVDEL+B3        ; move to b(B3)
                      M
                   01845          ADD32   HMOVVAL,MOVDEL
                      M
0464 6A66             M           MOVFP   HMOVVAL+B0,WREG       ; get lowest byte of a into w
0465 0FBB             M           ADDWF   MOVDEL+B0, F          ; add lowest byte of b, save in b(B0)
0466 6A67             M           MOVFP   HMOVVAL+B1,WREG       ; get 2nd byte of a into w
0467 11BC             M           ADDWFC  MOVDEL+B1, F          ; add 2nd byte of b, save in b(B1)
0468 6A68             M           MOVFP   HMOVVAL+B2,WREG       ; get 3rd byte of a into w
0469 11BD             M           ADDWFC  MOVDEL+B2, F          ; add 3rd byte of b, save in b(B2)
046A 6A69             M           MOVFP   HMOVVAL+B3,WREG       ; get 4th byte of a into w
046B 11BE             M           ADDWFC  MOVDEL+B3, F          ; add 4th byte of b, save in b(B3)
                      M
                   01846          SUB32   MOVPBUF,MOVDEL
                      M
046C 6AAF             M           MOVFP   MOVPBUF+B0,WREG       ; get lowest byte of a into w
046D 05BB             M           SUBWF   MOVDEL+B0, F          ; sub lowest byte of b, save in b(B0)
046E 6AB0             M           MOVFP   MOVPBUF+B1,WREG       ; get 2nd byte of a into w
046F 03BC             M           SUBWFB  MOVDEL+B1, F          ; sub 2nd byte of b, save in b(B1)
0470 6AB1             M           MOVFP   MOVPBUF+B2,WREG       ; get 3rd byte of a into w
0471 03BD             M           SUBWFB  MOVDEL+B2, F          ; sub 3rd byte of b, save in b(B2)
0472 6AB2             M           MOVFP   MOVPBUF+B3,WREG       ; get 4th byte of a into w
```

© 1997 Microchip Technology Inc.

```
0473 03BE            M          SUBWFB  MOVDEL+B3, F          ; sub 4th byte of b, save in b(B3)
                     M
0474 976C          01847        BTFSS   MOVSIGN,MSB
0475 C47F          01848        GOTO    mpos1
                   01849        NEG32   MOVDEL
                     M
0476 13BB            M          COMF    MOVDEL+B0, F
0477 13BC            M          COMF    MOVDEL+B1, F
0478 13BD            M          COMF    MOVDEL+B2, F
0479 13BE            M          COMF    MOVDEL+B3, F
047A 290A            M          CLRF    WREG, F
047B 15BB            M          INCF    MOVDEL+B0, F
047C 11BC            M          ADDWFC  MOVDEL+B1, F
047D 11BD            M          ADDWFC  MOVDEL+B2, F
047E 11BE            M          ADDWFC  MOVDEL+B3, F
                     M
047F               01850 mpos1
047F 97BE          01851        BTFSS   MOVDEL+B3,MSB
0480 C4E5          01852        GOTO    speedup               ; continue to speed up if in phase1
                   01853
                   01854        TFSZ16  T1                    ; if T1=0, maximum velocity not
                     M
0481 6A6D            M          MOVFP   T1+B0,WREG
0482 086E            M          IORWF   T1+B1,W
0483 330A            M          TSTFSZ  WREG
                   01855                                     ; reached, so T1=T2, otherwise T1
                   01856                                     ; has been set in speedup
0484 C4B8          01857        GOTO    t2net1
                   01858
                   01859        NEG32   A                     ; negate A for speeddown
                     M
0485 13A7            M          COMF    A+B0, F
0486 13A8            M          COMF    A+B1, F
0487 13A9            M          COMF    A+B2, F
0488 13AA            M          COMF    A+B3, F
0489 290A            M          CLRF    WREG, F
048A 15A7            M          INCF    A+B0, F
048B 11A8            M          ADDWFC  A+B1, F
048C 11A9            M          ADDWFC  A+B2, F
048D 11AA            M          ADDWFC  A+B3, F
                     M
                   01860        ADD32   MOVDEL,MOVTMP         ; test x-y < 0
                     M
048E 6ABB            M          MOVFP   MOVDEL+B0,WREG        ; get lowest byte of a into w
048F 0F1C            M          ADDWF   MOVTMP+B0, F          ; add lowest byte of b, save in b(B0)
0490 6ABC            M          MOVFP   MOVDEL+B1,WREG        ; get 2nd byte of a into w
0491 111D            M          ADDWFC  MOVTMP+B1, F          ; add 2nd byte of b, save in b(B1)
```


© 1997 Microchip Technology Inc.

```
0492 6ABD              M         MOVFP   MOVDEL+B2,WREG          ; get 3rd byte of a into w
0493 111E              M         ADDWFC  MOVTMP+B2, F            ; add 3rd byte of b, save in b(B2)
0494 6ABE              M         MOVFP   MOVDEL+B3,WREG          ; get 4th byte of a into w
0495 111F              M         ADDWFC  MOVTMP+B3, F            ; add 4th byte of b, save in b(B3)
                       M
0496 971F          01861         BTFSS   MOVTMP+B3,MSB           ; if new discretization error larger,
0497 C4AE          01862         GOTO    triok                   ; backup to define T2, otherwise ok
                   01863
0498 2B6F          01864         SETF    T2+B0, F                ; set T2=-1 for backup
0499 2B70          01865         SETF    T2+B1, F
                   01866         NEG32   A                       ; negate A to undo
                       M
049A 13A7              M         COMF    A+B0, F
049B 13A8              M         COMF    A+B1, F
049C 13A9              M         COMF    A+B2, F
049D 13AA              M         COMF    A+B3, F
049E 290A              M         CLRF    WREG, F
049F 15A7              M         INCF    A+B0, F
04A0 11A8              M         ADDWFC  A+B1, F
04A1 11A9              M         ADDWFC  A+B2, F
04A2 11AA              M         ADDWFC  A+B3, F
                       M
04A3 E5CA          01867         CALL    undoPosVel
                   01868         NEG32   A                       ; negate A again for speeddown
                       M
04A4 13A7              M         COMF    A+B0, F
04A5 13A8              M         COMF    A+B1, F
04A6 13A9              M         COMF    A+B2, F
04A7 13AA              M         COMF    A+B3, F
04A8 290A              M         CLRF    WREG, F
04A9 15A7              M         INCF    A+B0, F
04AA 11A8              M         ADDWFC  A+B1, F
04AB 11A9              M         ADDWFC  A+B2, F
04AC 11AA              M         ADDWFC  A+B3, F
                       M
04AD E5A8          01869         CALL    doPosVel                ; and reevaluate iterative equations
04AE               01870 triok
                   01871         ADD16   MOVTIME,T2              ; add time to T2
                       M
04AE 6A6A              M         MOVFP   MOVTIME+B0,WREG         ; get lowest byte of a into w
04AF 0F6F              M         ADDWF   T2+B0, F                ; add lowest byte of b, save in b(B0)
04B0 6A6B              M         MOVFP   MOVTIME+B1,WREG         ; get 2nd byte of a into w
04B1 1170              M         ADDWFC  T2+B1, F                ; add 2nd byte of b, save in b(B1)
                       M
                   01872         MOV16   T2,T1
                       M
04B2 6A6F              M         MOVFP   T2+B0,WREG              ; get byte of a into w
```

© 1997 Microchip Technology Inc.

```
04B3 4A6D              M          MOVPF    WREG,T1+B0              ; move to b(B0)
04B4 6A70              M          MOVFP    T2+B1,WREG              ; get byte of a into w
04B5 4A6E              M          MOVPF    WREG,T1+B1              ; move to b(B1)
                       M
04B6 1599          01873          INCF     MOVFLAG, F              ; increment move flag for phase2
04B7 C50E          01874          GOTO     mvok                    ; execute last phase1 move
                   01875
04B8               01876 t2net1
04B8 2B6F          01877          SETF     T2+B0, F                ; set T2=-1 for backup
04B9 2B70          01878          SETF     T2+B1, F
                   01879          ADD16    MOVTIME,T2              ; add time to T2
                       M
04BA 6A6A              M          MOVFP    MOVTIME+B0,WREG         ; get lowest byte of a into w
04BB 0F6F              M          ADDWF    T2+B0, F                ; add lowest byte of b, save in b(B0)
04BC 6A6B              M          MOVFP    MOVTIME+B1,WREG         ; get 2nd byte of a into w
04BD 1170              M          ADDWFC   T2+B1, F                ; add 2nd byte of b, save in b(B1)
                       M
                   01880
                   01881          MVFP32   MOVTMP,TMP              ; test if 3x-y < 0
                       M
04BE 781C              M          MOVFP    MOVTMP+B0,TMP+B0        ; move A(B0) to B(B0)
04BF 791D              M          MOVFP    MOVTMP+B1,TMP+B1        ; move A(B1) to B(B1)
04C0 7A1E              M          MOVFP    MOVTMP+B2,TMP+B2        ; move A(B2) to B(B2)
04C1 7B1F              M          MOVFP    MOVTMP+B3,TMP+B3        ; move A(B3) to B(B3)
                       M
                   01882          RLC32    MOVTMP
                       M
04C2 8804              M          BCF      _carry
04C3 1B1C              M          RLCF     MOVTMP+B0, F
04C4 1B1D              M          RLCF     MOVTMP+B1, F
04C5 1B1E              M          RLCF     MOVTMP+B2, F
04C6 1B1F              M          RLCF     MOVTMP+B3, F
                       M
                   01883          ADD32    TMP,MOVTMP
                       M
04C7 6A18              M          MOVFP    TMP+B0,WREG             ; get lowest byte of a into w
04C8 0F1C              M          ADDWF    MOVTMP+B0, F            ; add lowest byte of b, save in b(B0)
04C9 6A19              M          MOVFP    TMP+B1,WREG             ; get 2nd byte of a into w
04CA 111D              M          ADDWFC   MOVTMP+B1, F            ; add 2nd byte of b, save in b(B1)
04CB 6A1A              M          MOVFP    TMP+B2,WREG             ; get 3rd byte of a into w
04CC 111E              M          ADDWFC   MOVTMP+B2, F            ; add 3rd byte of b, save in b(B2)
04CD 6A1B              M          MOVFP    TMP+B3,WREG             ; get 4th byte of a into w
04CE 111F              M          ADDWFC   MOVTMP+B3, F            ; add 4th byte of b, save in b(B3)
                       M
                   01884          ADD32    MOVDEL,MOVTMP
                       M
04CF 6ABB              M          MOVFP    MOVDEL+B0,WREG          ; get lowest byte of a into w
```


© 1997 Microchip Technology Inc.

```
04D0 0F1C              M          ADDWF   MOVTMP+B0, F          ; add lowest byte of b, save in b(B0)
04D1 6ABC              M          MOVFP   MOVDEL+B1,WREG        ; get 2nd byte of a into w
04D2 111D              M          ADDWFC  MOVTMP+B1, F          ; add 2nd byte of b, save in b(B1)
04D3 6ABD              M          MOVFP   MOVDEL+B2,WREG        ; get 3rd byte of a into w
04D4 111E              M          ADDWFC  MOVTMP+B2, F          ; add 3rd byte of b, save in b(B2)
04D5 6ABE              M          MOVFP   MOVDEL+B3,WREG        ; get 4th byte of a into w
04D6 111F              M          ADDWFC  MOVTMP+B3, F          ; add 4th byte of b, save in b(B3)
                       M
04D7 971F          01885          BTFSS   MOVTMP+B3,MSB         ; if new discretization error larger,
04D8 C4DB          01886          GOTO    trapok                ; take one more flat step
04D9 2BBF          01887          SETF    PH2FLAT+B0, F
04DA 2BC0          01888          SETF    PH2FLAT+B1, F
04DB               01889 trapok
                   01890          ADD16   T2,PH2FLAT
                       M
04DB 6A6F              M          MOVFP   T2+B0,WREG            ; get lowest byte of a into w
04DC 0FBF              M          ADDWF   PH2FLAT+B0, F         ; add lowest byte of b, save in b(B0)
04DD 6A70              M          MOVFP   T2+B1,WREG            ; get 2nd byte of a into w
04DE 11C0              M          ADDWFC  PH2FLAT+B1, F         ; add 2nd byte of b, save in b(B1)
                       M
                   01891          SUB16   T1,PH2FLAT
                       M
04DF 6A6D              M          MOVFP   T1+B0,WREG            ; get lowest byte of a into w
04E0 05BF              M          SUBWF   PH2FLAT+B0, F         ; sub lowest byte of b, save in b(B0)
04E1 6A6E              M          MOVFP   T1+B1,WREG            ; get 2nd byte of a into w
04E2 03C0              M          SUBWFB  PH2FLAT+B1, F         ; sub 2nd byte of b, save in b(B1)
                       M
04E3 1599          01892          INCF    MOVFLAG, F            ; increment move flag for phase2
04E4 C50E          01893          GOTO    mvok                  ; execute last phase1 move
                   01894
04E5               01895 speedup
                   01896          MVFP32  V,MOVTMP              ; test if maximum velocity reached
                       M
04E5 7CAB              M          MOVFP   V+B0,MOVTMP+B0        ; move A(B0) to B(B0)
04E6 7DAC              M          MOVFP   V+B1,MOVTMP+B1        ; move A(B1) to B(B1)
04E7 7EAD              M          MOVFP   V+B2,MOVTMP+B2        ; move A(B2) to B(B2)
04E8 7FAE              M          MOVFP   V+B3,MOVTMP+B3        ; move A(B3) to B(B3)
                       M
                   01897          SUB32   MOVVBUF,MOVTMP
                       M
04E9 6AB3              M          MOVFP   MOVVBUF+B0,WREG       ; get lowest byte of a into w
04EA 051C              M          SUBWF   MOVTMP+B0, F          ; sub lowest byte of b, save in b(B0)
04EB 6AB4              M          MOVFP   MOVVBUF+B1,WREG       ; get 2nd byte of a into w
04EC 031D              M          SUBWFB  MOVTMP+B1, F          ; sub 2nd byte of b, save in b(B1)
04ED 6AB5              M          MOVFP   MOVVBUF+B2,WREG       ; get 3rd byte of a into w
04EE 031E              M          SUBWFB  MOVTMP+B2, F          ; sub 3rd byte of b, save in b(B2)
04EF 6AB6              M          MOVFP   MOVVBUF+B3,WREG       ; get 4th byte of a into w
```

© 1997 Microchip Technology Inc.

```
04F0 031F              M         SUBWFB  MOVTMP+B3, F          ; sub 4th byte of b, save in b(B3)
                       M
04F1 976C            01898       BTFSS   MOVSIGN,MSB
04F2 C4FC            01899       GOTO    mpos
                     01900       NEG32   MOVTMP
                       M
04F3 131C              M         COMF    MOVTMP+B0, F
04F4 131D              M         COMF    MOVTMP+B1, F
04F5 131E              M         COMF    MOVTMP+B2, F
04F6 131F              M         COMF    MOVTMP+B3, F
04F7 290A              M         CLRF    WREG, F
04F8 151C              M         INCF    MOVTMP+B0, F
04F9 111D              M         ADDWFC  MOVTMP+B1, F
04FA 111E              M         ADDWFC  MOVTMP+B2, F
04FB 111F              M         ADDWFC  MOVTMP+B3, F
                       M
04FC                 01901 mpos
04FC 971F            01902       BTFSS   MOVTMP+B3,MSB
04FD C50E            01903       GOTO    mvok                  ; if not, execute move
                     01904
                     01905       TFSZ16  T1                    ; if so, check to see if T1 has
                       M
04FE 6A6D              M         MOVFP   T1+B0,WREG
04FF 086E              M         IORWF   T1+B1,W
0500 330A              M         TSTFSZ  WREG
                     01906                                     ; already been set
0501 C50E            01907       GOTO    mvok
0502 E5CA            01908       CALL    undoPosVel            ; if not, backup and redo iterative
                     01909       CLR32   A                     ; equations, resulting in an actual
0503 29A7              M         CLRF    A+B0, F
0504 29A8              M         CLRF    A+B1, F
0505 29A9              M         CLRF    A+B2, F
0506 29AA              M         CLRF    A+B3, F
                       M
0507 E5A8            01910       CALL    doPosVel              ; maximum speed <= VL
0508 2B6D            01911       SETF    T1+B0, F              ; evaluate T1
0509 2B6E            01912       SETF    T1+B1, F
                     01913       ADD16   MOVTIME,T1
                       M
050A 6A6A              M         MOVFP   MOVTIME+B0,WREG       ; get lowest byte of a into w
050B 0F6D              M         ADDWF   T1+B0, F              ; add lowest byte of b, save in b(B0)
050C 6A6B              M         MOVFP   MOVTIME+B1,WREG       ; get 2nd byte of a into w
050D 116E              M         ADDWFC  T1+B1, F              ; add 2nd byte of b, save in b(B1)
                       M
050E                 01914 mvok
                     01915       MOV24   MOVPBUF+B1,POSITION   ; move Q8 calculated position to Q0 commanded position
                       M
```


© 1997 Microchip Technology Inc.

```
050E 6AB0              M          MOVFP   MOVPBUF+B1+B0,WREG      ; get byte of a into w
050F 4A58              M          MOVPF   WREG,POSITION+B0        ; move to b(B0)
0510 6AB1              M          MOVFP   MOVPBUF+B1+B1,WREG      ; get byte of a into w
0511 4A59              M          MOVPF   WREG,POSITION+B1        ; move to b(B1)
0512 6AB2              M          MOVFP   MOVPBUF+B1+B2,WREG      ; get byte of a into w
0513 4A5A              M          MOVPF   WREG,POSITION+B2        ; move to b(B2)
                       M
                     01916        MOV24   MOVVBUF+B0,VELOCITY     ; move Q0 calculated velocity to Q0 commanded velocity
                       M
0514 6AB3              M          MOVFP   MOVVBUF+B0+B0,WREG      ; get byte of a into w
0515 4A5B              M          MOVPF   WREG,VELOCITY+B0        ; move to b(B0)
0516 6AB4              M          MOVFP   MOVVBUF+B0+B1,WREG      ; get byte of a into w
0517 4A5C              M          MOVPF   WREG,VELOCITY+B1        ; move to b(B1)
0518 6AB5              M          MOVFP   MOVVBUF+B0+B2,WREG      ; get byte of a into w
0519 4A5D              M          MOVPF   WREG,VELOCITY+B2        ; move to b(B2)
                       M
051A 0002            01917        RETURN
                     01918
                     01919
051B                 01920 phase2
                     01921        TFSZ16  PH2FLAT                 ; is flat section finished?
                       M
051B 6ABF              M          MOVFP   PH2FLAT+B0,WREG
051C 08C0              M          IORWF   PH2FLAT+B1,W
051D 330A              M          TSTFSZ  WREG
051E C53F            01922        GOTO    flat
                     01923
                     01924        TFSZ32  MOVVBUF                 ; is velocity zero?
                       M
051F 6AB3              M          MOVFP   MOVVBUF+B0,WREG
0520 08B4              M          IORWF   MOVVBUF+B1,W
0521 08B5              M          IORWF   MOVVBUF+B2,W
0522 08B6              M          IORWF   MOVVBUF+B3,W
0523 330A              M          TSTFSZ  WREG
0524 C55C            01925        GOTO    mready                  ; if not, execute move
                     01926
0525 2999            01927        CLRF    MOVFLAG, F              ; if so, clear MOVFLAG
0526 8E98            01928        BCF     MOVSTAT,BIT6            ; clear motion status flag
0527 8D98            01929        BCF     MOVSTAT,BIT5            ; clear move in progress flag
                     01930        CLR32   A                       ; set zero velocity and acceleration,
0528 29A7              M          CLRF    A+B0, F
0529 29A8              M          CLRF    A+B1, F
052A 29A9              M          CLRF    A+B2, F
052B 29AA              M          CLRF    A+B3, F
                       M
                     01931        MOV16   MOVTIME,TAU
                       M
```

© 1997 Microchip Technology Inc.

```
052C 6A6A              M         MOVFP   MOVTIME+B0,WREG         ; get byte of a into w
052D 4A71              M         MOVPF   WREG,TAU+B0             ; move to b(B0)
052E 6A6B              M         MOVFP   MOVTIME+B1,WREG         ; get byte of a into w
052F 4A72              M         MOVPF   WREG,TAU+B1             ; move to b(B1)
                       M
                    01932        MOV32   OPOSITION,MOVPBUF       ; execute last move to P(0)+MOVVAL
                       M
0530 6A54              M         MOVFP   OPOSITION+B0,WREG       ; get byte of a into w
0531 4AAF              M         MOVPF   WREG,MOVPBUF+B0         ; move to b(B0)
0532 6A55              M         MOVFP   OPOSITION+B1,WREG       ; get byte of a into w
0533 4AB0              M         MOVPF   WREG,MOVPBUF+B1         ; move to b(B1)
0534 6A56              M         MOVFP   OPOSITION+B2,WREG       ; get byte of a into w
0535 4AB1              M         MOVPF   WREG,MOVPBUF+B2         ; move to b(B2)
0536 6A57              M         MOVFP   OPOSITION+B3,WREG       ; get byte of a into w
0537 4AB2              M         MOVPF   WREG,MOVPBUF+B3         ; move to b(B3)
                       M
                    01933        ADD24   MOVVAL,MOVPBUF+B1
                       M
0538 6A62              M         MOVFP   MOVVAL+B0,WREG          ; get lowest byte of a into w
0539 0FB0              M         ADDWF   MOVPBUF+B1+B0, F        ; add lowest byte of b, save in b(B0)
053A 6A63              M         MOVFP   MOVVAL+B1,WREG          ; get 2nd byte of a into w
053B 11B1              M         ADDWFC  MOVPBUF+B1+B1, F        ; add 2nd byte of b, save in b(B1)
053C 6A64              M         MOVFP   MOVVAL+B2,WREG          ; get 3rd byte of a into w
053D 11B2              M         ADDWFC  MOVPBUF+B1+B2, F        ; add 3rd byte of b, save in b(B2)
                       M
053E C55C           01934        GOTO    mready
                    01935
053F                01936 flat
053F 2B1C           01937        SETF    MOVTMP+B0, F
0540 2B1D           01938        SETF    MOVTMP+B1, F
                    01939        ADD16   MOVTMP,PH2FLAT          ; decrement by one use DEC16
                       M
0541 6A1C              M         MOVFP   MOVTMP+B0,WREG          ; get lowest byte of a into w
0542 0FBF              M         ADDWF   PH2FLAT+B0, F           ; add lowest byte of b, save in b(B0)
0543 6A1D              M         MOVFP   MOVTMP+B1,WREG          ; get 2nd byte of a into w
0544 11C0              M         ADDWFC  PH2FLAT+B1, F           ; add 2nd byte of b, save in b(B1)
                       M
                    01940
                    01941        TFSZ16  PH2FLAT
                       M
0545 6ABF              M         MOVFP   PH2FLAT+B0,WREG
0546 08C0              M         IORWF   PH2FLAT+B1,W
0547 330A              M         TSTFSZ  WREG
0548 C55C           01942        GOTO    mready
                    01943
0549 29AA           01944        CLRF    A+B3, F                 ; begin speed down section
                    01945        MOV24   AL,A
```


© 1997 Microchip Technology Inc.

```
                        M
054A 6A23               M           MOVFP   AL+B0,WREG              ; get byte of a into w
054B 4AA7               M           MOVPF   WREG,A+B0               ; move to b(B0)
054C 6A24               M           MOVFP   AL+B1,WREG              ; get byte of a into w
054D 4AA8               M           MOVPF   WREG,A+B1               ; move to b(B1)
054E 6A25               M           MOVFP   AL+B2,WREG              ; get byte of a into w
054F 4AA9               M           MOVPF   WREG,A+B2               ; move to b(B2)
                        M
0550 290A             01946         CLRF    WREG, F
0551 316C             01947         CPFSEQ  MOVSIGN
0552 C55C             01948         GOTO    mready
                      01949         NEG32   A
                        M
0553 13A7               M           COMF    A+B0, F
0554 13A8               M           COMF    A+B1, F
0555 13A9               M           COMF    A+B2, F
0556 13AA               M           COMF    A+B3, F
0557 290A               M           CLRF    WREG, F
0558 15A7               M           INCF    A+B0, F
0559 11A8               M           ADDWFC  A+B1, F
055A 11A9               M           ADDWFC  A+B2, F
055B 11AA               M           ADDWFC  A+B3, F
                        M
                      01950
055C                  01951 mready
                      01952         MOV24   MOVPBUF+B1,POSITION
                        M
055C 6AB0               M           MOVFP   MOVPBUF+B1+B0,WREG      ; get byte of a into w
055D 4A58               M           MOVPF   WREG,POSITION+B0        ; move to b(B0)
055E 6AB1               M           MOVFP   MOVPBUF+B1+B1,WREG      ; get byte of a into w
055F 4A59               M           MOVPF   WREG,POSITION+B1        ; move to b(B1)
0560 6AB2               M           MOVFP   MOVPBUF+B1+B2,WREG      ; get byte of a into w
0561 4A5A               M           MOVPF   WREG,POSITION+B2        ; move to b(B2)
                        M
                      01953         MOV24   MOVVBUF+B0,VELOCITY
                        M
0562 6AB3               M           MOVFP   MOVVBUF+B0+B0,WREG      ; get byte of a into w
0563 4A5B               M           MOVPF   WREG,VELOCITY+B0        ; move to b(B0)
0564 6AB4               M           MOVFP   MOVVBUF+B0+B1,WREG      ; get byte of a into w
0565 4A5C               M           MOVPF   WREG,VELOCITY+B1        ; move to b(B1)
0566 6AB5               M           MOVFP   MOVVBUF+B0+B2,WREG      ; get byte of a into w
0567 4A5D               M           MOVPF   WREG,VELOCITY+B2        ; move to b(B2)
                        M
0568 0002             01954         RETURN
                      01955
0569                  01956 vmove
                      01957         MVFP32  MOVVAL,MOVTMP           ; test if final velocity reached
```

© 1997 Microchip Technology Inc.

```
                    M
0569 7C62           M       MOVFP   MOVVAL+B0,MOVTMP+B0      ; move A(B0) to B(B0)
056A 7D63           M       MOVFP   MOVVAL+B1,MOVTMP+B1      ; move A(B1) to B(B1)
056B 7E64           M       MOVFP   MOVVAL+B2,MOVTMP+B2      ; move A(B2) to B(B2)
056C 7F65           M       MOVFP   MOVVAL+B3,MOVTMP+B3      ; move A(B3) to B(B3)
                    M
                 01958      SUB32   MOVVBUF,MOVTMP
                    M
056D 6AB3           M       MOVFP   MOVVBUF+B0,WREG          ; get lowest byte of a into w
056E 051C           M       SUBWF   MOVTMP+B0, F             ; sub lowest byte of b, save in b(B0)
056F 6AB4           M       MOVFP   MOVVBUF+B1,WREG          ; get 2nd byte of a into w
0570 031D           M       SUBWFB  MOVTMP+B1, F             ; sub 2nd byte of b, save in b(B1)
0571 6AB5           M       MOVFP   MOVVBUF+B2,WREG          ; get 3rd byte of a into w
0572 031E           M       SUBWFB  MOVTMP+B2, F             ; sub 3rd byte of b, save in b(B2)
0573 6AB6           M       MOVFP   MOVVBUF+B3,WREG          ; get 4th byte of a into w
0574 031F           M       SUBWFB  MOVTMP+B3, F             ; sub 4th byte of b, save in b(B3)
                    M
0575 976C        01959      BTFSS   MOVSIGN,MSB
0576 C580        01960      GOTO    vmpos
                 01961      NEG32   MOVTMP
                    M
0577 131C           M       COMF    MOVTMP+B0, F
0578 131D           M       COMF    MOVTMP+B1, F
0579 131E           M       COMF    MOVTMP+B2, F
057A 131F           M       COMF    MOVTMP+B3, F
057B 290A           M       CLRF    WREG, F
057C 151C           M       INCF    MOVTMP+B0, F
057D 111D           M       ADDWFC  MOVTMP+B1, F
057E 111E           M       ADDWFC  MOVTMP+B2, F
057F 111F           M       ADDWFC  MOVTMP+B3, F
                    M
0580             01962 vmpos
0580 971F        01963      BTFSS   MOVTMP+B3,MSB
0581 C59B        01964      GOTO    vmoveok                  ; if not, continue
                 01965
                 01966      CLR32   A                        ; if so, set A=0 and continue with
0582 29A7           M       CLRF    A+B0, F
0583 29A8           M       CLRF    A+B1, F
0584 29A9           M       CLRF    A+B2, F
0585 29AA           M       CLRF    A+B3, F
                    M
                 01967      MOV32   MOVVAL,MOVVBUF           ; move unless the final velocity
                    M
0586 6A62           M       MOVFP   MOVVAL+B0,WREG           ; get byte of a into w
0587 4AB3           M       MOVPF   WREG,MOVVEUF+B0          ; move to b(B0)
0588 6A63           M       MOVFP   MOVVAL+B1,WREG           ; get byte of a into w
0589 4AB4           M       MOVPF   WREG,MOVVEUF+B1          ; move to b(B1)
```


© 1997 Microchip Technology Inc.

```
058A 6A64                M          MOVFP   MOVVAL+B2,WREG          ; get byte of a into w
058B 4AB5                M          MOVPF   WREG,MOVVBUF+B2         ; move to b(B2)
058C 6A65                M          MOVFP   MOVVAL+B3,WREG          ; get byte of a into w
058D 4AB6                M          MOVPF   WREG,MOVVBUF+B3         ; move to b(B3)
                         M
                     01968                                         ; is zero.
058E 2999            01969          CLRF    MOVFLAG, F              ; clear MOVFLAG
058F 8D98            01970          BCF     MOVSTAT,BIT5            ; clear move in progress flag
                     01971          MOV16   MOVTIME,TAU
                         M
0590 6A6A                M          MOVFP   MOVTIME+B0,WREG         ; get byte of a into w
0591 4A71                M          MOVPF   WREG,TAU+B0             ; move to b(B0)
0592 6A6B                M          MOVFP   MOVTIME+B1,WREG         ; get byte of a into w
0593 4A72                M          MOVPF   WREG,TAU+B1             ; move to b(B1)
                         M
                     01972          TFSZ32  MOVVAL
                         M
0594 6A62                M          MOVFP   MOVVAL+B0,WREG
0595 0863                M          IORWF   MOVVAL+B1,W
0596 0864                M          IORWF   MOVVAL+B2,W
0597 0865                M          IORWF   MOVVAL+B3,W
0598 330A                M          TSTFSZ  WREG
0599 C59B            01973          GOTO    vmoveok
                     01974
059A 8E98            01975          BCF     MOVSTAT,BIT6            ; if final velocity is zero, clear
                     01976                                         ; motion status flag
059B                 01977 vmoveok
                     01978          MOV24   MOVPBUF+B1,POSITION
                         M
059B 6AB0                M          MOVFP   MOVPBUF+B1+B0,WREG      ; get byte of a into w
059C 4A58                M          MOVPF   WREG,POSITION+B0        ; move to b(B0)
059D 6AB1                M          MOVFP   MOVPBUF+B1+B1,WREG      ; get byte of a into w
059E 4A59                M          MOVPF   WREG,POSITION+B1        ; move to b(B1)
059F 6AB2                M          MOVFP   MOVPBUF+B1+B2,WREG      ; get byte of a into w
05A0 4A5A                M          MOVPF   WREG,POSITION+B2        ; move to b(B2)
                         M
                     01979          MOV24   MOVVBUF+B0,VELOCITY
                         M
05A1 6AB3                M          MOVFP   MOVVBUF+B0+B0,WREG      ; get byte of a into w
05A2 4A5B                M          MOVPF   WREG,VELOCITY+B0        ; move to b(B0)
05A3 6AB4                M          MOVFP   MOVVBUF+B0+B1,WREG      ; get byte of a into w
05A4 4A5C                M          MOVPF   WREG,VELOCITY+B1        ; move to b(B1)
05A5 6AB5                M          MOVFP   MOVVBUF+B0+B2,WREG      ; get byte of a into w
05A6 4A5D                M          MOVPF   WREG,VELOCITY+B2        ; move to b(B2)
                         M
05A7 0002            01980          RETURN
                     01981
```

© 1997 Microchip Technology Inc.

```
                    01982 ;*****************************************************************************
                    01983
                    01984 ;*****************************************************************************
                    01985 ; NAME:         doPosVel
                    01986 ;
                    01987 ; DESCRIPTION:  Evaluates the iterative equations for trapezoidal move
                    01988 ;               generation
                    01989 ;
                    01990 ;               V(k)=V(k-1)+A,          P(k)=P(k-1)+V(k-1)+A/2,
                    01991 ;
                    01992 ;               where abs(A)={AL,0} depending on the region of the trapezoid
                    01993 ;               being executed.
                    01994 ;
                    01995
05A8                01996 doPosVel
                    01997
                    01998         ADD32   MOVVBUF,MOVPBUF         ; P(k-1)+V(k-1)
                        M
05A8 6AB3               M         MOVFP   MOVVBUF+B0,WREG         ; get lowest byte of a into w
05A9 0FAF               M         ADDWF   MOVPBUF+B0, F           ; add lowest byte of b, save in b(B0)
05AA 6AB4               M         MOVFP   MOVVBUF+B1,WREG         ; get 2nd byte of a into w
05AB 11B0               M         ADDWFC  MOVPBUF+B1, F           ; add 2nd byte of b, save in b(B1)
05AC 6AB5               M         MOVFP   MOVVBUF+B2,WREG         ; get 3rd byte of a into w
05AD 11B1               M         ADDWFC  MOVPBUF+B2, F           ; add 3rd byte of b, save in b(B2)
05AE 6AB6               M         MOVFP   MOVVBUF+B3,WREG         ; get 4th byte of a into w
05AF 11B2               M         ADDWFC  MOVPBUF+B3, F           ; add 4th byte of b, save in b(B3)
                        M
                    01999         ADD32   A,MOVVBUF               ; V(k)=V(k-1)+A
                        M
05B0 6AA7               M         MOVFP   A+B0,WREG               ; get lowest byte of a into w
05B1 0FB3               M         ADDWF   MOVVBUF+B0, F           ; add lowest byte of b, save in b(B0)
05B2 6AA8               M         MOVFP   A+B1,WREG               ; get 2nd byte of a into w
05B3 11B4               M         ADDWFC  MOVVBUF+B1, F           ; add 2nd byte of b, save in b(B1)
05B4 6AA9               M         MOVFP   A+B2,WREG               ; get 3rd byte of a into w
05B5 11B5               M         ADDWFC  MOVVBUF+B2, F           ; add 3rd byte of b, save in b(B2)
05B6 6AAA               M         MOVFP   A+B3,WREG               ; get 4th byte of a into w
05B7 11B6               M         ADDWFC  MOVVBUF+B3, F           ; add 4th byte of b, save in b(B3)
                        M
                    02000
                    02001         MVFP32  A,MOVTMP                ; compute A/2
                        M
05B8 7CA7               M         MOVFP   A+B0,MOVTMP+B0          ; move A(B0) to B(B0)
05B9 7DA8               M         MOVFP   A+B1,MOVTMP+B1          ; move A(B1) to B(B1)
05BA 7EA9               M         MOVFP   A+B2,MOVTMP+B2          ; move A(B2) to B(B2)
05BB 7FAA               M         MOVFP   A+B3,MOVTMP+B3          ; move A(B3) to B(B3)
                        M
                    02002         RRC32   MOVTMP
```


© 1997 Microchip Technology Inc.

```
                    M
05BC 1A1F           M         RLCF    MOVTMP+B3,W           ; move sign into carry bit
05BD 191F           M         RRCF    MOVTMP+B3, F
05BE 191E           M         RRCF    MOVTMP+B2, F
05BF 191D           M         RRCF    MOVTMP+B1, F
05C0 191C           M         RRCF    MOVTMP+B0, F
                    M
                  02003
                  02004         ADD32   MOVTMP,MOVPBUF        ; P(k)=P(k-1)+V(k-1)+A/2,
                    M
05C1 6A1C           M         MOVFP   MOVTMP+B0,WREG        ; get lowest byte of a into w
05C2 0FAF           M         ADDWF   MOVPBUF+B0, F         ; add lowest byte of b, save in b(B0)
05C3 6A1D           M         MOVFP   MOVTMP+B1,WREG        ; get 2nd byte of a into w
05C4 11B0           M         ADDWFC  MOVPBUF+B1, F         ; add 2nd byte of b, save in b(B1)
05C5 6A1E           M         MOVFP   MOVTMP+B2,WREG        ; get 3rd byte of a into w
05C6 11B1           M         ADDWFC  MOVPBUF+B2, F         ; add 3rd byte of b, save in b(B2)
05C7 6A1F           M         MOVFP   MOVTMP+B3,WREG        ; get 4th byte of a into w
05C8 11B2           M         ADDWFC  MOVPBUF+B3, F         ; add 4th byte of b, save in b(B3)
                    M
                  02005
05C9 0002         02006         RETURN
                  02007
                  02008 ;*******************************************************************************
                  02009
                  02010 ;*******************************************************************************
                  02011 ; NAME: undoPosVel
                  02012 ;
                  02013 ; DESCRIPTION:  Backward iteration of the equations for trapezoidal move
                  02014 ;               generation
                  02015 ;
                  02016 ;               V(k-1)=V(k)-A,          P(k-1)=P(k)-V(k-1)-A/2,
                  02017 ;
                  02018 ;               where abs(A)={AL,0} depending on the region of the trapezoid
                  02019 ;               being executed. This routine is used to reverse a step about
                  02020 ;               to be made beyond a decision point.
                  02021 ;
                  02022
05CA              02023 undoPosVel
                  02024
                  02025         SUB32   A,MOVVBUF             ; V(k-1)=V(k)-A
                    M
05CA 6AA7           M         MOVFP   A+B0,WREG             ; get lowest byte of a into w
05CB 05B3           M         SUBWF   MOVVBUF+B0, F         ; sub lowest byte of b, save in b(B0)
05CC 6AA8           M         MOVFP   A+B1,WREG             ; get 2nd byte of a into w
05CD 03B4           M         SUBWFB  MOVVBUF+B1, F         ; sub 2nd byte of b, save in b(B1)
05CE 6AA9           M         MOVFP   A+B2,WREG             ; get 3rd byte of a into w
05CF 03B5           M         SUBWFB  MOVVBUF+B2, F         ; sub 3rd byte of b, save in b(B2)
```

© 1997 Microchip Technology Inc.

```
05D0 6AAA              M         MOVFP   A+B3,WREG              ; get 4th byte of a into w
05D1 03B6              M         SUBWFB  MOVVBUF+B3, F          ; sub 4th byte of b, save in b(B3)
                       M
                     02026       SUB32   MOVVBUF,MOVPBUF        ; P(k)-V(k-1)
                       M
05D2 6AB3              M         MOVFP   MOVVBUF+B0,WREG        ; get lowest byte of a into w
05D3 05AF              M         SUBWF   MOVPBUF+B0, F          ; sub lowest byte of b, save in b(B0)
05D4 6AB4              M         MOVFP   MOVVBUF+B1,WREG        ; get 2nd byte of a into w
05D5 03B0              M         SUBWFB  MOVPBUF+B1, F          ; sub 2nd byte of b, save in b(B1)
05D6 6AB5              M         MOVFP   MOVVBUF+B2,WREG        ; get 3rd byte of a into w
05D7 03B1              M         SUBWFB  MOVPBUF+B2, F          ; sub 3rd byte of b, save in b(B2)
05D8 6AB6              M         MOVFP   MOVVBUF+B3,WREG        ; get 4th byte of a into w
05D9 03B2              M         SUBWFB  MOVPBUF+B3, F          ; sub 4th byte of b, save in b(B3)
                       M
                     02027
                     02028       MVFP32  A,MOVTMP               ; compute A/2
                       M
05DA 7CA7              M         MOVFP   A+B0,MOVTMP+B0         ; move A(B0) to B(B0)
05DB 7DA8              M         MOVFP   A+B1,MOVTMP+B1         ; move A(B1) to B(B1)
05DC 7EA9              M         MOVFP   A+B2,MOVTMP+B2         ; move A(B2) to B(B2)
05DD 7FAA              M         MOVFP   A+B3,MOVTMP+B3         ; move A(B3) to B(B3)
                       M
                     02029       RRC32   MOVTMP
                       M
05DE 1A1F              M         RLCF    MOVTMP+B3,W            ; move sign into carry bit
05DF 191F              M         RRCF    MOVTMP+B3, F
05E0 191E              M         RRCF    MOVTMP+B2, F
05E1 191D              M         RRCF    MOVTMP+B1, F
05E2 191C              M         RRCF    MOVTMP+B0, F
                       M
                     02030
                     02031       SUB32   MOVTMP,MOVPBUF         ; P(k-1)=P(k)-V(k-1)-A/2,
                       M
05E3 6A1C              M         MOVFP   MOVTMP+B0,WREG         ; get lowest byte of a into w
05E4 05AF              M         SUBWF   MOVPBUF+B0, F          ; sub lowest byte of b, save in b(B0)
05E5 6A1D              M         MOVFP   MOVTMP+B1,WREG         ; get 2nd byte of a into w
05E6 03B0              M         SUBWFB  MOVPBUF+B1, F          ; sub 2nd byte of b, save in b(B1)
05E7 6A1E              M         MOVFP   MOVTMP+B2,WREG         ; get 3rd byte of a into w
05E8 03B1              M         SUBWFB  MOVPBUF+B2, F          ; sub 3rd byte of b, save in b(B2)
05E9 6A1F              M         MOVFP   MOVTMP+B3,WREG         ; get 4th byte of a into w
05EA 03B2              M         SUBWFB  MOVPBUF+B3, F          ; sub 4th byte of b, save in b(B3)
                       M
                     02032
05EB 0002            02033       RETURN
                     02034
                     02035 ;*****************************************************************************
                     02036
```


© 1997 Microchip Technology Inc.

```
                    02037 ;*******************************************************************************
                    02038 ; NAME:          doMPosMVel
                    02039 ;
                    02040 ; DESCRIPTION:  Calculates current position from UpCount and DownCount
                    02041 ;
                    02042
05EC                02043 doMPosMVel
                    02044
                    02045 ; Do UpCounter first
                    02046
                    02047          MVFP16  UPCOUNT,TMP+B0          ; save old upcount
                        M
05EC 78B7               M          MOVFP   UPCOUNT+B0,TMP+B0+B0    ; move A(B0) to B(B0)
05ED 79B8               M          MOVFP   UPCOUNT+B1,TMP+B0+B1    ; move A(B1) to B(B1)
                        M
05EE                02048 readUp
05EE 4C0A           02049          MOVPF   TMR0H,WREG
05EF 4BB7           02050          MOVPF   TMR0L,UPCOUNT+B0
05F0 310C           02051          CPFSEQ  TMR0H                   ; Skip next if HI hasn't changed
05F1 C5EE           02052          GOTO    readUp                  ; HI changed, re-read LO
05F2 4AB8           02053          MOVPF   WREG,UPCOUNT+B1         ; OK to store HI now
                    02054
05F3 2978           02055          CLRF    MVELOCITY+B0, F         ; clear bits below binary point
                    02056
                    02057          MOV16   UPCOUNT,MVELOCITY+B1    ; compute upcount increment
                        M
05F4 6AB7               M          MOVFP   UPCOUNT+B0,WREG         ; get byte of a into w
05F5 4A79               M          MOVPF   WREG,MVELOCITY+B1+B0    ; move to b(B0)
05F6 6AB8               M          MOVFP   UPCOUNT+B1,WREG         ; get byte of a into w
05F7 4A7A               M          MOVPF   WREG,MVELOCITY+B1+B1    ; move to b(B1)
                        M
                    02058          SUB16   TMP+B0,MVELOCITY+B1
                        M
05F8 6A18               M          MOVFP   TMP+B0+B0,WREG          ; get lowest byte of a into w
05F9 0579               M          SUBWF   MVELOCITY+B1+B0, F      ; sub lowest byte of b, save in b(B0)
05FA 6A19               M          MOVFP   TMP+B0+B1,WREG          ; get 2nd byte of a into w
05FB 037A               M          SUBWFB  MVELOCITY+B1+B1, F      ; sub 2nd byte of b, save in b(B1)
                        M
                    02059
                    02060 ; Now do DownCounter
                    02061
                    02062          MVFP16  DOWNCOUNT,TMP+B0        ; save old downcount
                        M
05FC 78B9               M          MOVFP   DOWNCOUNT+B0,TMP+B0+B0  ; move A(B0) to B(B0)
05FD 79BA               M          MOVFP   DOWNCOUNT+B1,TMP+B0+B1  ; move A(B1) to B(B1)
                        M
05FE                02063 readDown
```

© 1997 Microchip Technology Inc.

```
05FE B802  02064         MOVLB   BANK2                   ; timers in Bank 2
05FF 530A  02065         MOVPF   TMR3H,WREG
0600 52B9  02066         MOVPF   TMR3L,DOWNCOUNT+B0
0601 3113  02067         CPFSEQ  TMR3H                   ; Skip next if HI hasn't changed
0602 C5FE  02068         GOTO    readDown                ; HI changed, re-read LO
0603 4ABA  02069         MOVPF   WREG,DOWNCOUNT+B1       ; OK to store HI now
           02070
           02071         MVFP16  DOWNCOUNT+B0,TMP+B2     ; compute downcount increment
               M
0604 7AB9      M         MOVFP   DOWNCOUNT+B0+B0,TMP+B2+B0 ; move A(B0) to B(B0)
0605 7BBA      M         MOVFP   DOWNCOUNT+B0+B1,TMP+B2+B1 ; move A(B1) to B(B1)
               M
           02072         SUB16   TMP+B0,TMP+B2
               M
0606 6A18      M         MOVFP   TMP+B0+B0,WREG          ; get lowest byte of a into w
0607 051A      M         SUBWF   TMP+B2+B0, F            ; sub lowest byte of b, save in b(B0)
0608 6A19      M         MOVFP   TMP+B0+B1,WREG          ; get 2nd byte of a into w
0609 031B      M         SUBWFB  TMP+B2+B1, F            ; sub 2nd byte of b, save in b(B1)
               M
           02073
           02074         SUB16   TMP+B2,MVELOCITY+B1     ; compute new measured velocity
               M
060A 6A1A      M         MOVFP   TMP+B2+B0,WREG          ; get lowest byte of a into w
060B 0579      M         SUBWF   MVELOCITY+B1+B0, F      ; sub lowest byte of b, save in b(B0)
060C 6A1B      M         MOVFP   TMP+B2+B1,WREG          ; get 2nd byte of a into w
060D 037A      M         SUBWFB  MVELOCITY+B1+B1, F      ; sub 2nd byte of b, save in b(B1)
               M
           02075
060E 297B  02076         CLRF    MVELOCITY+B3, F         ; sign extend measured velocity for
060F 9F7A  02077         BTFSC   MVELOCITY+B2,MSB        ; 24 bit addition to measured position
0610 2B7B  02078         SETF    MVELOCITY+B3, F
           02079
           02080
           02081
           02082         ADD24   MVELOCITY+B1,MPOSITION  ; compute new measured position
               M
0611 6A79      M         MOVFP   MVELOCITY+B1+B0,WREG    ; get lowest byte of a into w
0612 0F75      M         ADDWF   MPOSITION+B0, F         ; add lowest byte of b, save in b(B0)
0613 6A7A      M         MOVFP   MVELOCITY+B1+B1,WREG    ; get 2nd byte of a into w
0614 1176      M         ADDWFC  MPOSITION+B1, F         ; add 2nd byte of b, save in b(B1)
0615 6A7B      M         MOVFP   MVELOCITY+B1+B2,WREG    ; get 3rd byte of a into w
0616 1177      M         ADDWFC  MPOSITION+B2, F         ; add 3rd byte of b, save in b(B2)
               M
           02083                                         ; delta position = measured velocity
           02084
0617 0002  02085         RETURN
           02086
```


© 1997 Microchip Technology Inc.

```
                    02087 ;*******************************************************************************
                    02088
                    02089 ;*******************************************************************************
                    02090 ; NAME:         doIntegral
                    02091 ;
                    02092 ; DESCRIPTION:  Evaluates the integral for the servo calculations.
                    02093 ;
0618                02094 doIntegral
                    02095
                    02096         ADD16   U0,INTEGRAL             ; do integral
                        M
0618 6A8F               M         MOVFP   U0+B0,WREG              ; get lowest byte of a into w
0619 0F9B               M         ADDWF   INTEGRAL+B0, F          ; add lowest byte of b, save in b(B0)
061A 6A90               M         MOVFP   U0+B1,WREG              ; get 2nd byte of a into w
061B 119C               M         ADDWFC  INTEGRAL+B1, F          ; add 2nd byte of b, save in b(B1)
                        M
                    02097
061C 0002           02098         RETURN
                    02099
                    02100 ;*******************************************************************************
                    02101
                    02102 ;*******************************************************************************
                    02103 ; NAME:         doExtstat
                    02104 ;
                    02105 ; DESCRIPTION:  Get +limit,-limit,GPI from PORTB and set in EXTSTAT
                    02106 ;
061D                02107 doExtstat
061D 9407           02108         BTFSS   _intir
061E C627           02109         GOTO    otherbits
                    02110         MOV24   MPOSITION,INDEXPOS
                        M
061F 6A75               M         MOVFP   MPOSITION+B0,WREG       ; get byte of a into w
0620 4AC1               M         MOVPF   WREG,INDEXPOS+B0        ; move to b(B0)
0621 6A76               M         MOVFP   MPOSITION+B1,WREG       ; get byte of a into w
0622 4AC2               M         MOVPF   WREG,INDEXPOS+B1        ; move to b(B1)
0623 6A77               M         MOVFP   MPOSITION+B2,WREG       ; get byte of a into w
0624 4AC3               M         MOVPF   WREG,INDEXPOS+B2        ; move to b(B2)
                        M
0625 8C07           02111         BCF     _intir
0626 8797           02112         BSF     EXTSTAT,MSB
                    02113
0627                02114 otherbits
0627 B800           02115         MOVLB   BANK0                   ; get +limit,-limit and GPI
0628 6A12           02116         MOVFP   PORTB,WREG
0629 190A           02117         RRCF    WREG, F                 ; arrange in correct bit positions
062A B561           02118         ANDLW   0x61
062B 4A18           02119         MOVPF   WREG,TMP
```

© 1997 Microchip Technology Inc.

```
062C 1D18          02120         SWAPF   TMP, F
062D 0818          02121         IORWF   TMP,W
062E 0997          02122         IORWF   EXTSTAT, F              ; set in EXTSTAT
                   02123
062F 0002          02124         RETURN
                   02125
                   02126 ;*************************************************************************
                   02127
                   02128 ;*************************************************************************
                   02129 ; NAME:          Dmult
                   02130 ;
                   02131 ; DESCRIPTION:  Mult: AARG (16 bits) * BARG (16 bits) -> DPX (32 bits)
                   02132 ;
                   02133 ;       (a) Load the 1st operand in locations AARG+B0 & AARG+B1 (16 bits)
                   02134 ;       (b) Load the 2nd operand in locations BARG+B0 & BARG+B1 (16 bits)
                   02135 ;       (c) CALL Dmult
                   02136 ;       (d) The 32 bit result is in locations (DPX+B0,DPX+B1,DPX+B2,DPX+B3)
                   02137 ;
                   02138 ;       In the signed case, a savings of 9 clks can be realized by choosing
                   02139 ;       BARG as the positive factor in the product when possible.
                   02140 ;
                   02141 ; TIMING (worst case):  unsigned:                    173 clks
                   02142 ;                       signed:    if BARG positive:  170 clks
                   02143 ;                                  if BARG negative:  179 clks
                   02144 ;
                   02145
                   02146 ;*************************************************************************
                   02147
00000001           02148 SIGNED  equ     TRUE            ; Set This To 'TRUE' for signed multiply
                   02149                                 ; and 'FALSE' for unsigned.
                   02150 ;*************************************************************************
                   02151 ;               Multiplication Macro
                   02152 ;*************************************************************************
                   02153 ;
                   02154 ; TIMING:       unsigned:     11+7*10+8*11   = 169 clks
                   02155 ;(worst case)   signed:       11+7*10+7*11+5 = 163 clks
                   02156 ;
                   02157 MULTMAC  MACRO
                   02158         variable i
                   02159
                   02160           variable i = 0
                   02161           #if   SIGNED
                   02162              variable MULT_LP_CNT = 15
                   02163           #else
                   02164              variable MULT_LP_CNT = 16
                   02165           #endif
                   02166              .while i < MULT_LP_CNT
```


© 1997 Microchip Technology Inc.

```
02167
02168              .if i < 8
02169                 BTFSC      BARG+B0,i         ; test low byte
02170              .else
02171                 BTFSC      BARG+B1,i-8       ; test high byte
02172              .fi
02173                 GOTO    add#v(i)
02174              .if i < 8
02175                 RLCF    DPX+B3,W          ; rotate sign into carry bit
02176                 RRCF    DPX+B3, F         ; for i < 8, no meaningful bits
02177                 RRCF    DPX+B2, F         ; are in DPX+B0
02178                 RRCF    DPX+B1, F
02179              .else
02180                 RLCF    DPX+B3,W          ; rotate sign into carry bit
02181                 RRCF    DPX+B3, F
02182                 RRCF    DPX+B2, F
02183                 RRCF    DPX+B1, F
02184                 RRCF    DPX+B0, F
02185              .fi
02186                 variable i = i+1
02187              .endw
02188
02189
02190                 CLRF    DPX+B0, F         ; if we get here, BARG = 0
02191                 RETURN
02192
02193
02194
02195 add0
02196                 MOVFP   AARG+B0,WREG
02197                 ADDWF   DPX+B2, F         ;add lsb
02198                 MOVFP   AARG+B1,WREG
02199                 ADDWFC  DPX+B3, F         ;add msb
02200                 RLCF    AARG+B1,W         ; rotate sign into carry bit
02201                 RRCF    DPX+B3, F         ; for i < 8, no meaningful bits
02202                 RRCF    DPX+B2, F         ; are in DPX+B0
02203                 RRCF    DPX+B1, F
02204
02205           variable i = 1
02206
02207
02208              .while i < MULT_LP_CNT
02209
02210              .if i < 8
02211                 BTFSS   BARG+B0,i         ; test low byte
02212              .else
02213                 BTFSS   BARG+B1,i-8       ; test high byte
```

© 1997 Microchip Technology Inc.

```
02214                .fi
02215                   GOTO    noadd#v(i)
02216 add#v(i)
02217                   MOVFP   AARG+B0,WREG
02218                   ADDWF   DPX+B2, F         ;add lsb
02219                   MOVFP   AARG+B1,WREG
02220                   ADDWFC  DPX+B3, F         ;add msb
02221
02222 noadd#v(i)
02223                .if i < 8
02224
02225                   RLCF    AARG+B1,W         ; rotate sign into carry bit
02226                   RRCF    DPX+B3, F         ; for i < 8, no meaningful bits
02227                   RRCF    DPX+B2, F         ; are in DPX+B0
02228                   RRCF    DPX+B1, F
02229
02230                .else
02231
02232                   RLCF    AARG+B1,W         ; rotate sign into carry bit
02233                   RRCF    DPX+B3, F
02234                   RRCF    DPX+B2, F
02235                   RRCF    DPX+B1, F
02236                   RRCF    DPX+B0, F
02237
02238                .fi
02239
02240                variable i = i+1
02241                .endw
02242
02243             #if   SIGNED
02244
02245                   RLCF    AARG+B1,W         ; since BARG is always made positive,
02246                   RRCF    DPX+B3, F         ; the last bit is known to be zero.
02247                   RRCF    DPX+B2, F
02248                   RRCF    DPX+B1, F
02249                   RRCF    DPX+B0, F
02250
02251             #endif
02252
02253           ENDM
02254
02255 ;            Double Precision Multiply ( 16x16 -> 32 )
02256 ;           ( AARG*BARG -> : 32 bit output in DPX
02257 ;
0630  02258 Dmult
02259        #if   SIGNED
02260
```


© 1997 Microchip Technology Inc.

```
0630 971F          02261         BTFSS   BARG+B1,MSB             ; test sign of BARG
0631 C63C          02262         GOTO    argsok                  ; if positive, ok
                   02263         NEG16   AARG+B0                 ; if negative, then negate both
                       M
0632 131C              M         COMF    AARG+B0+B0, F
0633 131D              M         COMF    AARG+B0+B1, F
0634 290A              M         CLRF    WREG, F
0635 151C              M         INCF    AARG+B0+B0, F
0636 111D              M         ADDWFC  AARG+B0+B1, F
                       M
                   02264         NEG16   BARG+B0                 ; AARG and BARG
                       M
0637 131E              M         COMF    BARG+B0+B0, F
0638 131F              M         COMF    BARG+B0+B1, F
0639 290A              M         CLRF    WREG, F
063A 151E              M         INCF    BARG+B0+B0, F
063B 111F              M         ADDWFC  BARG+B0+B1, F
                       M
                   02265
                   02266     #endif
063C               02267 argsok
063C 291B          02268         CLRF    DPX+B3, F               ; clear initial partial product
063D 291A          02269         CLRF    DPX+B2, F
                   02270
                   02271         MULTMAC                         ; use macro for multiplication
  0000                 M         variable i
                       M
  0000                 M           variable i = 0
                       M           #if   SIGNED
  000F                 M              variable MULT_LP_CNT = 15
                       M           #else
                       M              variable MULT_LP_CNT = 16
                       M           #endif
                       M              .while i < MULT_LP_CNT
                       M
                       M              .if i < 8
063E 981E              M                 BTFSC      BARG+B0,i        ; test low byte
                       M              .else
                       M                 BTFSC      BARG+B1,i-8      ; test high byte
                       M              .fi
063F C6A1              M                 GOTO    add0
                       M              .if i < 8
0640 1A1B              M                 RLCF    DPX+B3,W            ; rotate sign into carry bit
0641 191B              M                 RRCF    DPX+B3, F           ; for i < 8, no meaningful bits
0642 191A              M                 RRCF    DPX+B2, F           ; are in DPX+B0
0643 1919              M                 RRCF    DPX+B1, F
                       M              .else
```

© 1997 Microchip Technology Inc.

```
                    M                 RLCF    DPX+B3,W          ; rotate sign into carry bit
                    M                 RRCF    DPX+B3, F
                    M                 RRCF    DPX+B2, F
                    M                 RRCF    DPX+B1, F
                    M                 RRCF    DPX+B0, F
                    M               .fi
      0001          M                 variable i = i+1
                    M
                    M               .if i < 8
0644 991E           M                 BTFSC     BARG+B0,i        ; test low byte
                    M               .else
                    M                 BTFSC     BARG+B1,i-8      ; test high byte
                    M               .fi
0645 C6AB           M                 GOTO    add1
                    M               .if i < 8
0646 1A1B           M                 RLCF    DPX+B3,W          ; rotate sign into carry bit
0647 191B           M                 RRCF    DPX+B3, F         ; for i < 8, no meaningful bits
0648 191A           M                 RRCF    DPX+B2, F         ; are in DPX+B0
0649 1919           M                 RRCF    DPX+B1, F
                    M               .else
                    M                 RLCF    DPX+B3,W          ; rotate sign into carry bit
                    M                 RRCF    DPX+B3, F
                    M                 RRCF    DPX+B2, F
                    M                 RRCF    DPX+B1, F
                    M                 RRCF    DPX+B0, F
                    M               .fi
      0002          M                 variable i = i+1
                    M
                    M               .if i < 8
064A 9A1E           M                 BTFSC   BARG+B0,i         ; test low byte
                    M               .else
                    M                 BTFSC   BARG+B1,i-8       ; test high byte
                    M               .fi
064B C6B5           M                 GOTO    add2
                    M               .if i < 8
064C 1A1B           M                 RLCF    DPX+B3,W          ; rotate sign into carry bit
064D 191B           M                 RRCF    DPX+B3, F         ; for i < 8, no meaningful bits
064E 191A           M                 RRCF    DPX+B2, F         ; are in DPX+B0
064F 1919           M                 RRCF    DPX+B1, F
                    M               .else
                    M                 RLCF    DPX+B3,W          ; rotate sign into carry bit
                    M                 RRCF    DPX+B3, F
                    M                 RRCF    DPX+B2, F
                    M                 RRCF    DPX+B1, F
                    M                 RRCF    DPX+B0, F
                    M               .fi
      0003          M                 variable i = i+1
```


© 1997 Microchip Technology Inc.

```
                    M
                    M           .if i < 8
0650 9B1E           M              BTFSC   BARG+B0,i        ; test low byte
                    M           .else
                    M              BTFSC   BARG+B1,i-8      ; test high byte
                    M           .fi
0651 C6BF           M              GOTO    add3
                    M           .if i < 8
0652 1A1B           M              RLCF    DPX+B3,W         ; rotate sign into carry bit
0653 191B           M              RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
0654 191A           M              RRCF    DPX+B2, F        ; are in DPX+B0
0655 1919           M              RRCF    DPX+B1, F
                    M           .else
                    M              RLCF    DPX+B3,W         ; rotate sign into carry bit
                    M              RRCF    DPX+B3, F
                    M              RRCF    DPX+B2, F
                    M              RRCF    DPX+B1, F
                    M              RRCF    DPX+B0, F
                    M           .fi
  0004              M              variable i = i+1
                    M
                    M           .if i < 8
0656 9C1E           M              BTFSC   BARG+B0,i        ; test low byte
                    M           .else
                    M              BTFSC   BARG+B1,i-8      ; test high byte
                    M           .fi
0657 C6C9           M              GOTO    add4
                    M           .if i < 8
0658 1A1B           M              RLCF    DPX+B3,W         ; rotate sign into carry bit
0659 191B           M              RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
065A 191A           M              RRCF    DPX+B2, F        ; are in DPX+B0
065B 1919           M              RRCF    DPX+B1, F
                    M           .else
                    M              RLCF    DPX+B3,W         ; rotate sign into carry bit
                    M              RRCF    DPX+B3, F
                    M              RRCF    DPX+B2, F
                    M              RRCF    DPX+B1, F
                    M              RRCF    DPX+B0, F
                    M           .fi
  0005              M              variable i = i+1
                    M
                    M           .if i < 8
065C 9D1E           M              BTFSC   BARG+B0,i        ; test low byte
                    M           .else
                    M              BTFSC   BARG+B1,i-8      ; test high byte
                    M           .fi
065D C6D3           M              GOTO    add5
```

© 1997 Microchip Technology Inc.

```
                    M               .if i < 8
065E 1A1B           M                  RLCF    DPX+B3,W        ; rotate sign into carry bit
065F 191B           M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
0660 191A           M                  RRCF    DPX+B2, F       ; are in DPX+B0
0661 1919           M                  RRCF    DPX+B1, F
                    M               .else
                    M                  RLCF    DPX+B3,W        ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F
                    M                  RRCF    DPX+B2, F
                    M                  RRCF    DPX+B1, F
                    M                  RRCF    DPX+B0, F
                    M               .fi
  0006              M                  variable i = i+1
                    M
                    M               .if i < 8
0662 9E1E           M                  BTFSC   BARG+B0,i       ; test low byte
                    M               .else
                    M                  BTFSC   BARG+B1,i-8     ; test high byte
                    M               .fi
0663 C6DD           M                  GOTO    add6
                    M               .if i < 8
0664 1A1B           M                  RLCF    DPX+B3,W        ; rotate sign into carry bit
0665 191B           M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
0666 191A           M                  RRCF    DPX+B2, F       ; are in DPX+B0
0667 1919           M                  RRCF    DPX+B1, F
                    M               .else
                    M                  RLCF    DPX+B3,W        ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F
                    M                  RRCF    DPX+B2, F
                    M                  RRCF    DPX+B1, F
                    M                  RRCF    DPX+B0, F
                    M               .fi
  0007              M                  variable i = i+1
                    M
                    M               .if i < 8
0668 9F1E           M                  BTFSC   BARG+B0,i       ; test low byte
                    M               .else
                    M                  BTFSC   BARG+B1,i-8     ; test high byte
                    M               .fi
0669 C6E7           M                  GOTO    add7
                    M               .if i < 8
066A 1A1B           M                  RLCF    DPX+B3,W        ; rotate sign into carry bit
066B 191B           M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
066C 191A           M                  RRCF    DPX+B2, F       ; are in DPX+B0
066D 1919           M                  RRCF    DPX+B1, F
                    M               .else
                    M                  RLCF    DPX+B3,W        ; rotate sign into carry bit
```


© 1997 Microchip Technology Inc.

```
                    M               RRCF    DPX+B3, F
                    M               RRCF    DPX+B2, F
                    M               RRCF    DPX+B1, F
                    M               RRCF    DPX+B0, F
                    M             .fi
  0008              M               variable i = i+1
                    M
                    M             .if i < 8
                    M               BTFSC   BARG+B0,i       ; test low byte
                    M             .else
066E 981F           M               BTFSC   BARG+B1,i-8     ; test high byte
                    M             .fi
066F C6F1           M               GOTO    add8
                    M             .if i < 8
                    M               RLCF    DPX+B3,W        ; rotate sign into carry bit
                    M               RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                    M               RRCF    DPX+B2, F       ; are in DPX+B0
                    M               RRCF    DPX+B1, F
                    M             .else
0670 1A1B           M               RLCF    DPX+B3,W        ; rotate sign into carry bit
0671 191B           M               RRCF    DPX+B3, F
0672 191A           M               RRCF    DPX+B2, F
0673 1919           M               RRCF    DPX+B1, F
0674 1918           M               RRCF    DPX+B0, F
                    M             .fi
  0009              M               variable i = i+1
                    M
                    M             .if i < 8
                    M               BTFSC   BARG+B0,i       ; test low byte
                    M             .else
0675 991F           M               BTFSC   BARG+B1,i-8     ; test high byte
                    M             .fi
0676 C6FC           M               GOTO    add9
                    M             .if i < 8
                    M               RLCF    DPX+B3,W        ; rotate sign into carry bit
                    M               RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                    M               RRCF    DPX+B2, F       ; are in DPX+B0
                    M               RRCF    DPX+B1, F
                    M             .else
0677 1A1B           M               RLCF    DPX+B3,W        ; rotate sign into carry bit
0678 191B           M               RRCF    DPX+B3, F
0679 191A           M               RRCF    DPX+B2, F
067A 1919           M               RRCF    DPX+B1, F
067B 1918           M               RRCF    DPX+B0, F
                    M             .fi
  000A              M               variable i = i+1
                    M
```

© 1997 Microchip Technology Inc.

```
                    M               .if i < 8
                    M                 BTFSC   BARG+B0,i       ; test low byte
                    M               .else
067C 9A1F           M                 BTFSC   BARG+B1,i-8     ; test high byte
                    M               .fi
067D C707           M                 GOTO    add10
                    M               .if i < 8
                    M                 RLCF    DPX+B3,W        ; rotate sign into carry bit
                    M                 RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                    M                 RRCF    DPX+B2, F       ; are in DPX+B0
                    M                 RRCF    DPX+B1, F
                    M               .else
067E 1A1B           M                 RLCF    DPX+B3,W        ; rotate sign into carry bit
067F 191B           M                 RRCF    DPX+B3, F
0680 191A           M                 RRCF    DPX+B2, F
0681 1919           M                 RRCF    DPX+B1, F
0682 1918           M                 RRCF    DPX+B0, F
                    M               .fi
  000B              M                 variable i = i+1
                    M
                    M               .if i < 8
                    M                 BTFSC   BARG+B0,i       ; test low byte
                    M               .else
0683 9B1F           M                 BTFSC   BARG+B1,i-8     ; test high byte
                    M               .fi
0684 C712           M                 GOTO    add11
                    M               .if i < 8
                    M                 RLCF    DPX+B3,W        ; rotate sign into carry bit
                    M                 RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                    M                 RRCF    DPX+B2, F       ; are in DPX+B0
                    M                 RRCF    DPX+B1, F
                    M               .else
0685 1A1B           M                 RLCF    DPX+B3,W        ; rotate sign into carry bit
0686 191B           M                 RRCF    DPX+B3, F
0687 191A           M                 RRCF    DPX+B2, F
0688 1919           M                 RRCF    DPX+B1, F
0689 1918           M                 RRCF    DPX+B0, F
                    M               .fi
  000C              M                 variable i = i+1
                    M
                    M               .if i < 8
                    M                 BTFSC   BARG+B0,i       ; test low byte
                    M               .else
068A 9C1F           M                 BTFSC   BARG+B1,i-8     ; test high byte
                    M               .fi
068B C71D           M                 GOTO    add12
                    M               .if i < 8
```


© 1997 Microchip Technology Inc.

```
                M               RLCF    DPX+B3,W        ; rotate sign into carry bit
                M               RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                M               RRCF    DPX+B2, F       ; are in DPX+B0
                M               RRCF    DPX+B1, F
                M             .else
068C 1A1B       M               RLCF    DPX+B3,W        ; rotate sign into carry bit
068D 191B       M               RRCF    DPX+B3, F
068E 191A       M               RRCF    DPX+B2, F
068F 1919       M               RRCF    DPX+B1, F
0690 1918       M               RRCF    DPX+B0, F
                M             .fi
  000D          M               variable i = i+1
                M
                M             .if i < 8
                M               BTFSC   BARG+B0,i       ; test low byte
                M             .else
0691 9D1F       M               BTFSC   BARG+B1,i-8     ; test high byte
                M             .fi
0692 C728       M               GOTO    add13
                M             .if i < 8
                M               RLCF    DPX+B3,W        ; rotate sign into carry bit
                M               RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                M               RRCF    DPX+B2, F       ; are in DPX+B0
                M               RRCF    DPX+B1, F
                M             .else
0693 1A1B       M               RLCF    DPX+B3,W        ; rotate sign into carry bit
0694 191B       M               RRCF    DPX+B3, F
0695 191A       M               RRCF    DPX+B2, F
0696 1919       M               RRCF    DPX+B1, F
0697 1918       M               RRCF    DPX+B0, F
                M             .fi
  000E          M               variable i = i+1
                M
                M             .if i < 8
                M               BTFSC   BARG+B0,i       ; test low byte
                M             .else
0698 9E1F       M               BTFSC   BARG+B1,i-8     ; test high byte
                M             .fi
0699 C733       M               GOTO    add14
                M             .if i < 8
                M               RLCF    DPX+B3,W        ; rotate sign into carry bit
                M               RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                M               RRCF    DPX+B2, F       ; are in DPX+B0
                M               RRCF    DPX+B1, F
                M             .else
069A 1A1B       M               RLCF    DPX+B3,W        ; rotate sign into carry bit
069B 191B       M               RRCF    DPX+B3, F
```

© 1997 Microchip Technology Inc.

```
069C 191A          M                RRCF    DPX+B2, F
069D 1919          M                RRCF    DPX+B1, F
069E 1918          M                RRCF    DPX+B0, F
                   M             .fi
  000F             M                variable i = i+1
                   M             .endw
                   M
                   M
069F 2918          M                CLRF    DPX+B0, F      ; if we get here, BARG = 0
06A0 0002          M                RETURN
                   M
                   M
                   M
06A1               M add0
06A1 6A1C          M                MOVFP   AARG+B0,WREG
06A2 0F1A          M                ADDWF   DPX+B2, F       ;add lsb
06A3 6A1D          M                MOVFP   AARG+B1,WREG
06A4 111B          M                ADDWFC  DPX+B3, F       ;add msb
06A5 1A1D          M                RLCF    AARG+B1,W       ; rotate sign into carry bit
06A6 191B          M                RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
06A7 191A          M                RRCF    DPX+B2, F       ; are in DPX+B0
06A8 1919          M                RRCF    DPX+B1, F
                   M
  0001             M          variable i = 1
                   M
                   M
                   M             .while i < MULT_LP_CNT
                   M
                   M             .if i < 8
06A9 911E          M                BTFSS   BARG+B0,i       ; test low byte
                   M             .else
                   M                BTFSS   BARG+B1,i-8     ; test high byte
                   M             .fi
06AA C6AF          M                GOTO    noadd1
06AB               M add1
06AB 6A1C          M                MOVFP   AARG+B0,WREG
06AC 0F1A          M                ADDWF   DPX+B2, F       ; add lsb
06AD 6A1D          M                MOVFP   AARG+B1,WREG
06AE 111B          M                ADDWFC  DPX+B3, F       ; add msb
                   M
06AF               M noadd1
                   M             .if i < 8
                   M
06AF 1A1D          M                RLCF    AARG+B1,W       ; rotate sign into carry bit
06B0 191B          M                RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
06B1 191A          M                RRCF    DPX+B2, F       ; are in DPX+B0
06B2 1919          M                RRCF    DPX+B1, F
```


© 1997 Microchip Technology Inc.

```
                    M
                    M               .else
                    M
                    M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F
                    M                  RRCF    DPX+B2, F
                    M                  RRCF    DPX+B1, F
                    M                  RRCF    DPX+B0, F
                    M
                    M               .fi
                    M
  0002              M               variable i = i+1
                    M
                    M               .if i < 8
06B3 921E           M                  BTFSS   BARG+B0,i       ; test low byte
                    M               .else
                    M                  BTFSS   BARG+B1,i-8     ; test high byte
                    M               .fi
06B4 C6B9           M                  GOTO    noadd2
06B5                M add2
06B5 6A1C           M                  MOVFP   AARG+B0,WREG
06B6 0F1A           M                  ADDWF   DPX+B2, F       ;add lsb
06B7 6A1D           M                  MOVFP   AARG+B1,WREG
06B8 111B           M                  ADDWFC  DPX+B3, F       ;add msb
                    M
06B9                M noadd2
                    M               .if i < 8
                    M
06B9 1A1D           M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
06BA 191B           M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
06BB 191A           M                  RRCF    DPX+B2, F       ; are in DPX+B0
06BC 1919           M                  RRCF    DPX+B1, F
                    M
                    M               .else
                    M
                    M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F
                    M                  RRCF    DPX+B2, F
                    M                  RRCF    DPX+B1, F
                    M                  RRCF    DPX+B0, F
                    M
                    M               .fi
                    M
  0003              M               variable i = i+1
                    M
                    M               .if i < 8
06BD 931E           M                  BTFSS   BARG+B0,i       ; test low byte
```


© 1997 Microchip Technology Inc.

```
                    M               .else
                    M                  BTFSS   BARG+B1,i-8      ; test high byte
                    M               .fi
06BE C6C3           M                  GOTO    noadd3
06BF                M add3
06BF 6A1C           M                  MOVFP   AARG+B0,WREG
06C0 0F1A           M                  ADDWF   DPX+B2, F         ;add lsb
06C1 6A1D           M                  MOVFP   AARG+B1,WREG
06C2 111B           M                  ADDWFC  DPX+B3, F         ;add msb
                    M
06C3                M noadd3
                    M               .if i < 8
                    M
06C3 1A1D           M                  RLCF    AARG+B1,W         ; rotate sign into carry bit
06C4 191B           M                  RRCF    DPX+B3, F         ; for i < 8, no meaningful bits
06C5 191A           M                  RRCF    DPX+B2, F         ; are in DPX+B0
06C6 1919           M                  RRCF    DPX+B1, F
                    M
                    M               .else
                    M
                    M                  RLCF    AARG+B1,W         ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F
                    M                  RRCF    DPX+B2, F
                    M                  RRCF    DPX+B1, F
                    M                  RRCF    DPX+B0, F
                    M
                    M               .fi
                    M
  0004              M               variable i = i+1
                    M
                    M               .if i < 8
06C7 941E           M                  BTFSS   BARG+B0,i         ; test low byte
                    M               .else
                    M                  BTFSS   BARG+B1,i-8       ; test high byte
                    M               .fi
06C8 C6CD           M                  GOTO    noadd4
06C9                M add4
06C9 6A1C           M                  MOVFP   AARG+B0,WREG
06CA 0F1A           M                  ADDWF   DPX+B2, F         ;add lsb
06CB 6A1D           M                  MOVFP   AARG+B1,WREG
06CC 111B           M                  ADDWFC  DPX+B3, F         ;add msb
                    M
06CD                M noadd4
                    M               .if i < 8
                    M
06CD 1A1D           M                  RLCF    AARG+B1,W        ; rotate sign into carry bit
06CE 191B           M                  RRCF    DPX+B3, F        ; for i < 8, no meaningful bits
```


© 1997 Microchip Technology Inc.

```
06CF 191A           M                RRCF    DPX+B2, F       ; are in DPX+B0
06D0 1919           M                RRCF    DPX+B1, F
                    M
                    M             .else
                    M
                    M                RLCF    AARG+B1,W       ; rotate sign into carry bit
                    M                RRCF    DPX+B3, F
                    M                RRCF    DPX+B2, F
                    M                RRCF    DPX+B1, F
                    M                RRCF    DPX+B0, F
                    M
                    M             .fi
                    M
  0005              M             variable i = i+1
                    M
                    M             .if i < 8
06D1 951E           M                BTFSS   BARG+B0,i        ; test low byte
                    M             .else
                    M                BTFSS   BARG+B1,i-8      ; test high byte
                    M             .fi
06D2 C6D7           M                GOTO    noadd5
06D3                M add5
06D3 6A1C           M                MOVFP   AARG+B0,WREG
06D4 0F1A           M                ADDWF   DPX+B2, F        ;add lsb
06D5 6A1D           M                MOVFP   AARG+B1,WREG
06D6 111B           M                ADDWFC  DPX+B3, F        ;add msb
                    M
06D7                M noadd5
                    M             .if i < 8
                    M
06D7 1A1D           M                RLCF    AARG+B1,W       ; rotate sign into carry bit
06D8 191B           M                RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
06D9 191A           M                RRCF    DPX+B2, F       ; are in DPX+B0
06DA 1919           M                RRCF    DPX+B1, F
                    M
                    M             .else
                    M
                    M                RLCF    AARG+B1,W       ; rotate sign into carry bit
                    M                RRCF    DPX+B3, F
                    M                RRCF    DPX+B2, F
                    M                RRCF    DPX+B1, F
                    M                RRCF    DPX+B0, F
                    M
                    M             .fi
                    M
  0006              M             variable i = i+1
                    M
```

© 1997 Microchip Technology Inc.

```
                    M               .if i < 8
06DB 961E           M                  BTFSS   BARG+B0,i        ; test low byte
                    M               .else
                    M                  BTFSS   BARG+B1,i-8      ; test high byte
                    M               .fi
06DC C6E1           M                  GOTO    noadd6
06DD                M add6
06DD 6A1C           M                  MOVFP   AARG+B0,WREG
06DE 0F1A           M                  ADDWF   DPX+B2, F        ;add lsb
06DF 6A1D           M                  MOVFP   AARG+B1,WREG
06E0 111B           M                  ADDWFC  DPX+B3, F        ;add msb
                    M
06E1                M noadd6
                    M               .if i < 8
                    M
06E1 1A1D           M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
06E2 191B           M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
06E3 191A           M                  RRCF    DPX+B2, F       ; are in DPX+B0
06E4 1919           M                  RRCF    DPX+B1, F
                    M
                    M               .else
                    M
                    M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F
                    M                  RRCF    DPX+B2, F
                    M                  RRCF    DPX+B1, F
                    M                  RRCF    DPX+B0, F
                    M
                    M               .fi
                    M
  0007              M               variable i = i+1
                    M
                    M               .if i < 8
06E5 971E           M                  BTFSS   BARG+B0,i        ; test low byte
                    M               .else
                    M                  BTFSS   BARG+B1,i-8      ; test high byte
                    M               .fi
06E6 C6EB           M                  GOTO    noadd7
06E7                M add7
06E7 6A1C           M                  MOVFP   AARG+B0,WREG
06E8 0F1A           M                  ADDWF   DPX+B2, F        ;add lsb
06E9 6A1D           M                  MOVFP   AARG+B1,WREG
06EA 111B           M                  ADDWFC  DPX+B3, F        ;add msb
                    M
06EB                M noadd7
                    M               .if i < 8
                    M
```


© 1997 Microchip Technology Inc.

```
06EB 1A1D               M                RLCF    AARG+B1,W       ; rotate sign into carry bit
06EC 191B               M                RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
06ED 191A               M                RRCF    DPX+B2, F       ; are in DPX+B0
06EE 1919               M                RRCF    DPX+B1, F
                        M
                        M             .else
                        M
                        M                RLCF    AARG+B1,W       ; rotate sign into carry bit
                        M                RRCF    DPX+B3, F
                        M                RRCF    DPX+B2, F
                        M                RRCF    DPX+B1, F
                        M                RRCF    DPX+B0, F
                        M
                        M             .fi
                        M
  0008                  M             variable i = i+1
                        M
                        M             .if i < 8
                        M                BTFSS   BARG+B0,i       ; test low byte
                        M             .else
06EF 901F               M                BTFSS   BARG+B1,i-8     ; test high byte
                        M             .fi
06F0 C6F5               M                GOTO    noadd8
06F1                    M add8
06F1 6A1C               M                MOVFP   AARG+B0,WREG
06F2 0F1A               M                ADDWF   DPX+B2, F       ;add lsb
06F3 6A1D               M                MOVFP   AARG+B1,WREG
06F4 111B               M                ADDWFC  DPX+B3, F       ;add msb
                        M
06F5                    M noadd8
                        M             .if i < 8
                        M
                        M                RLCF    AARG+B1,W       ; rotate sign into carry bit
                        M                RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                        M                RRCF    DPX+B2, F       ; are in DPX+B0
                        M                RRCF    DPX+B1, F
                        M
                        M             .else
                        M
06F5 1A1D               M                RLCF    AARG+B1,W       ; rotate sign into carry bit
06F6 191B               M                RRCF    DPX+B3, F
06F7 191A               M                RRCF    DPX+B2, F
06F8 1919               M                RRCF    DPX+B1, F
06F9 1918               M                RRCF    DPX+B0, F
                        M
                        M             .fi
                        M
```

© 1997 Microchip Technology Inc.

```
  0009                  M               variable i = i+1
                        M
                        M               .if i < 8
                        M                  BTFSS   BARG+B0,i        ; test low byte
                        M               .else
06FA 911F               M                  BTFSS   BARG+B1,i-8      ; test high byte
                        M               .fi
06FB C700               M                  GOTO    noadd9
06FC                    M add9
06FC 6A1C               M                  MOVFP   AARG+B0,WREG
06FD 0F1A               M                  ADDWF   DPX+B2, F        ;add lsb
06FE 6A1D               M                  MOVFP   AARG+B1,WREG
06FF 111B               M                  ADDWFC  DPX+B3, F        ;add msb
                        M
0700                    M noadd9
                        M               .if i < 8
                        M
                        M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                        M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                        M                  RRCF    DPX+B2, F       ; are in DPX+B0
                        M                  RRCF    DPX+B1, F
                        M
                        M               .else
                        M
0700 1A1D               M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
0701 191B               M                  RRCF    DPX+B3, F
0702 191A               M                  RRCF    DPX+B2, F
0703 1919               M                  RRCF    DPX+B1, F
0704 1918               M                  RRCF    DPX+B0, F
                        M
                        M               .fi
                        M
  000A                  M               variable i = i+1
                        M
                        M               .if i < 8
                        M                  BTFSS      BARG+B0,i        ; test low byte
                        M               .else
0705 921F               M                  BTFSS      BARG+B1,i-8      ; test high byte
                        M               .fi
0706 C70B               M                  GOTO    noadd10
0707                    M add10
0707 6A1C               M                  MOVFP   AARG+B0,WREG
0708 0F1A               M                  ADDWF   DPX+B2, F        ;add lsb
0709 6A1D               M                  MOVFP   AARG+B1,WREG
070A 111B               M                  ADDWFC  DPX+B3, F        ;add msb
                        M
070B                    M noadd10
```


© 1997 Microchip Technology Inc.

```
                    M               .if i < 8
                    M
                    M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                    M                  RRCF    DPX+B2, F       ; are in DPX+B0
                    M                  RRCF    DPX+B1, F
                    M
                    M               .else
                    M
070B 1A1D           M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
070C 191B           M                  RRCF    DPX+B3, F
070D 191A           M                  RRCF    DPX+B2, F
070E 1919           M                  RRCF    DPX+B1, F
070F 1918           M                  RRCF    DPX+B0, F
                    M
                    M               .fi
                    M
  000B              M               variable i = i+1
                    M
                    M               .if i < 8
                    M                  BTFSS   BARG+B0,i       ; test low byte
                    M               .else
0710 931F           M                  BTFSS   BARG+B1,i-8     ; test high byte
                    M               .fi
0711 C716           M                  GOTO    noadd11
0712                M add11
0712 6A1C           M                  MOVFP   AARG+B0,WREG
0713 0F1A           M                  ADDWF   DPX+B2, F       ; add lsb
0714 6A1D           M                  MOVFP   AARG+B1,WREG
0715 111B           M                  ADDWFC  DPX+B3, F       ; add msb
                    M
0716                M noadd11
                    M               .if i < 8
                    M
                    M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                    M                  RRCF    DPX+B2, F       ; are in DPX+B0
                    M                  RRCF    DPX+B1, F
                    M
                    M               .else
                    M
0716 1A1D           M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
0717 191B           M                  RRCF    DPX+B3, F
0718 191A           M                  RRCF    DPX+B2, F
0719 1919           M                  RRCF    DPX+B1, F
071A 1918           M                  RRCF    DPX+B0, F
                    M
```

© 1997 Microchip Technology Inc.

```
                    M              .fi
                    M
  000C              M              variable i = i+1
                    M
                    M              .if i < 8
                    M                 BTFSS   BARG+B0,i        ; test low byte
                    M              .else
071B 941F           M                 BTFSS   BARG+B1,i-8      ; test high byte
                    M              .fi
071C C721           M                 GOTO    noadd12
071D                M add12
071D 6A1C           M                 MOVFP   AARG+B0,WREG
071E 0F1A           M                 ADDWF   DPX+B2, F        ; add lsb
071F 6A1D           M                 MOVFP   AARG+B1,WREG
0720 111B           M                 ADDWFC  DPX+B3, F        ; add msb
                    M
0721                M noadd12
                    M              .if i < 8
                    M
                    M                 RLCF    AARG+B1,W      ; rotate sign into carry bit
                    M                 RRCF    DPX+B3, F      ; for i < 8, no meaningful bits
                    M                 RRCF    DPX+B2, F      ; are in DPX+B0
                    M                 RRCF    DPX+B1, F
                    M
                    M              .else
                    M
0721 1A1D           M                 RLCF    AARG+B1,W      ; rotate sign into carry bit
0722 191B           M                 RRCF    DPX+B3, F
0723 191A           M                 RRCF    DPX+B2, F
0724 1919           M                 RRCF    DPX+B1, F
0725 1918           M                 RRCF    DPX+B0, F
                    M
                    M              .fi
                    M
  000D              M              variable i = i+1
                    M
                    M              .if i < 8
                    M                 BTFSS   BARG+B0,i        ; test low byte
                    M              .else
0726 951F           M                 BTFSS   BARG+B1,i-8      ; test high byte
                    M              .fi
0727 C72C           M                 GOTO    noadd13
0728                M add13
0728 6A1C           M                 MOVFP   AARG+B0,WREG
0729 0F1A           M                 ADDWF   DPX+B2, F        ; add lsb
072A 6A1D           M                 MOVFP   AARG+B1,WREG
072B 111B           M                 ADDWFC  DPX+B3, F        ; add msb
```


© 1997 Microchip Technology Inc.

```
                    M
072C                M noadd13
                    M               .if i < 8
                    M
                    M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                    M                  RRCF    DPX+B2, F       ; are in DPX+B0
                    M                  RRCF    DPX+B1, F
                    M
                    M               .else
                    M
072C 1A1D           M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
072D 191B           M                  RRCF    DPX+B3, F
072E 191A           M                  RRCF    DPX+B2, F
072F 1919           M                  RRCF    DPX+B1, F
0730 1918           M                  RRCF    DPX+B0, F
                    M
                    M               .fi
                    M
  000E              M               variable i = i+1
                    M
                    M               .if i < 8
                    M                  BTFSS   BARG+B0,i        ; test low byte
                    M               .else
0731 961F           M                  BTFSS   BARG+B1,i-8      ; test high byte
                    M               .fi
0732 C737           M                  GOTO    noadd14
0733                M add14
0733 6A1C           M                  MOVFP   AARG+B0,WREG
0734 0F1A           M                  ADDWF   DPX+B2, F        ;add lsb
0735 6A1D           M                  MOVFP   AARG+B1,WREG
0736 111B           M                  ADDWFC  DPX+B3, F        ;add msb
                    M
0737                M noadd14
                    M               .if i < 8
                    M
                    M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
                    M                  RRCF    DPX+B3, F       ; for i < 8, no meaningful bits
                    M                  RRCF    DPX+B2, F       ; are in DPX+B0
                    M                  RRCF    DPX+B1, F
                    M
                    M               .else
                    M
0737 1A1D           M                  RLCF    AARG+B1,W       ; rotate sign into carry bit
0738 191B           M                  RRCF    DPX+B3, F
0739 191A           M                  RRCF    DPX+B2, F
073A 1919           M                  RRCF    DPX+B1, F
```

© 1997 Microchip Technology Inc.

```
073B 1918              M                RRCF    DPX+B0, F
                       M
                       M             .fi
                       M
  000F                 M             variable i = i+1
                       M             .endw
                       M
                       M          #if   SIGNED
                       M
073C 1A1D              M                RLCF    AARG+B1,W       ; since BARG is always made positive,
073D 191B              M                RRCF    DPX+B3, F       ; the last bit is known to be zero.
073E 191A              M                RRCF    DPX+B2, F
073F 1919              M                RRCF    DPX+B1, F
0740 1918              M                RRCF    DPX+B0, F
                       M
                       M          #endif
                       M
                    02272
0741 0002           02273        RETURN
                    02274
                    02275 ;*************************************************************************
                    02276 ;
                    02277
                    02278 ;*************************************************************************
                    02279 ; NAME:         doCaptureRegs
                    02280 ;
                    02281 ; DESCRIPTION:  Captures Desired Register Values To PIC-MASTER Trace Buffer
                    02282 ;               Intended for PICMASTER Demo/debug/servo tuning Purposes Only
                    02283 ;               Capture The following registers to Trace Buffer by putting
                    02284 ;               A Trace point on a TABLW instruction. Trace only 2nd Cycle
                    02285 ;
                    02286 ;               (a) POSERROR (position error : 16 bits)
                    02287 ;               (b) VELERROR (velocity error : 16 bits)
                    02288 ;               (c) MPOSITION (measured position value : 24 bits)
                    02289 ;               (d) MVELOCITY (measured velocity value : 24 bits)
                    02290 ;               (e) POSITION (commanded position : 24 bits)
                    02291 ;               (f) VELOCITY (commanded velocity : 24 bits)
                    02292 ;               (g) Y (output of servo loop : 32 bits)
                    02293 ;               (h) YPWM (output value written to PWM : 10 bits)
                    02294 ;
                    02295 ;
                    02296 #define CaptureAddr     0x8000
                    02297 ;
0742                02298 doCaptureRegs
                    02299                                       ; !end! hdr !skip start!
0742 B000           02300        movlw   (CaptureAddr & 0xff)
0743 010D           02301        movwf   TBLPTRL
```


© 1997 Microchip Technology Inc.

```
0744 B080          02302         movlw   CaptureAddr/256
0745 010E          02303         movwf   TBLPTRH                ; setup table pointer address
                   02304
0746 AC7C          02305         tablwt  0,0,POSERROR+B0 ; dummy tablwt
0747 A67D          02306         tlwt    1,POSERROR+B1           ; now table latch = 16 bits contents of POSERROR
0748               02307 capPerr
0748 AC7C          02308         tablwt  0,0,POSERROR+B0 ; perform actual table write of POSERROR
                   02309
0749 AC7F          02310         tablwt  0,0,VELERROR+B0
074A A680          02311         tlwt    1,VELERROR+B1           ; capture Velocity error
074B               02312 capVerr
074B AC7F          02313         tablwt  0,0,VELERROR+B0
                   02314
074C AC75          02315         tablwt  0,0,MPOSITION+B0
074D A676          02316         tlwt    1,MPOSITION+B1          ; capture measured position
074E               02317 capMpos
074E AC75          02318         tablwt  0,0,MPOSITION+B0
                   02319
074F AC58          02320         tablwt  0,0,POSITION+B0
0750 A659          02321         tlwt    1,POSITION+B1           ; capture commanded position
0751               02322 capPos
0751 AC58          02323         tablwt  0,0,POSITION+B0
                   02324
0752 AC78          02325         tablwt  0,0,MVELOCITY+B0
0753 A679          02326         tlwt    1,MVELOCITY+B1          ; capture measured velocity
0754               02327 capMvel
0754 AC78          02328         tablwt  0,0,MVELOCITY+B0
                   02329
0755 AC5B          02330         tablwt  0,0,VELOCITY+B0
0756 A65C          02331         tlwt    1,VELOCITY+B1           ; capture commanded velocity
0757               02332 capVel
0757 AC5B          02333         tablwt  0,0,VELOCITY+B0
                   02334
                   02335         DEC16   CAPTMP
                       M
0758 290A              M         CLRF    WREG, F
0759 07C8              M         DECF    CAPTMP+B0, F
075A 03C9              M         SUBWFB  CAPTMP+B1, F
                       M
                   02336         TFSZ16  CAPTMP
                       M
075B 6AC8              M         MOVFP   CAPTMP+B0,WREG
075C 08C9              M         IORWF   CAPTMP+B1,W
075D 330A              M         TSTFSZ  WREG
075E 0002          02337         RETURN
075F 29C5          02338         CLRF    CAPFLAG, F
                   02339         MOV16   CAPCOUNT,CAPTMP
```

© 1997 Microchip Technology Inc.

```
                        M
0760 6AC6               M           MOVFP   CAPCOUNT+B0,WREG        ; get byte of a into w
0761 4AC8               M           MOVPF   WREG,CAPTMP+B0          ; move to b(B0)
0762 6AC7               M           MOVFP   CAPCOUNT+B1,WREG        ; get byte of a into w
0763 4AC9               M           MOVPF   WREG,CAPTMP+B1          ; move to b(B1)
                        M
0764 0001           02340           HALT
0765 0002           02341           RETURN
                    02342
                    02343
                    02344 ;*****************************************************************************
                    02345 ;
                    02346 ;
                    02347 ;       TABLES:
                    02348
                    02349
                    02350           CMD_START CMD_TABLE
                        M
0766                    M CMD_TABLE
                    02351           CMD_DEF do_null,DO_NULL
                        M
0766 000D               M           DATA    DO_NULL
0767 00A6               M           DATA    do_null
                    02352           CMD_DEF do_move,DO_MOVE
                        M
0768 004D               M           DATA    DO_MOVE
0769 00A8               M           DATA    do_move
                    02353           CMD_DEF do_mode,DO_MODE
                        M
076A 004F               M           DATA    DO_MODE
076B 00B6               M           DATA    do_mode
                    02354           CMD_DEF do_setparameter,DO_SETPARAMETER
                        M
076C 0053               M           DATA    DO_SETPARAMETER
076D 00D2               M           DATA    do_setparameter
                    02355           CMD_DEF do_readparameter,DO_READPARAMETER
                        M
076E 0052               M           DATA    DO_READPARAMETER
076F 00F9               M           DATA    do_readparameter
                    02356           CMD_DEF do_shutter,DO_SHUTTER
                        M
0770 0043               M           DATA    DO_SHUTTER
0771 0123               M           DATA    do_shutter
                    02357           CMD_DEF do_readcomposition,DO_READCOMPOSITION
                        M
0772 0050               M           DATA    DO_READCOMPOSITION
0773 0145               M           DATA    do_readcomposition
```


© 1997 Microchip Technology Inc.

```
                    02358          CMD_DEF do_readcomvelocity,DO_READCOMVELOCITY
                        M
0774 0056               M          DATA    DO_READCOMVELOCITY
0775 014E               M          DATA    do_readcomvelocity
                    02359          CMD_DEF do_readactposition,DO_READACTPOSITION
                        M
0776 0070               M          DATA    DO_READACTPOSITION
0777 0157               M          DATA    do_readactposition
                    02360          CMD_DEF do_readactvelocity,DO_READACTVELOCITY
                        M
0778 0076               M          DATA    DO_READACTVELOCITY
0779 0160               M          DATA    do_readactvelocity
                    02361          CMD_DEF do_externalstatus,DO_EXTERNALSTATUS
                        M
077A 0058               M          DATA    DO_EXTERNALSTATUS
077B 0169               M          DATA    do_externalstatus
                    02362          CMD_DEF do_movestatus,DO_MOVESTATUS
                        M
077C 0059               M          DATA    DO_MOVESTATUS
077D 0170               M          DATA    do_movestatus
                    02363          CMD_DEF do_readindposition,DO_READINDPOSITION
                        M
077E 0049               M          DATA    DO_READINDPOSITION
077F 0174               M          DATA    do_readindposition
                    02364          CMD_DEF do_setposition,DO_SETPOSITION
                        M
0780 0048               M          DATA    DO_SETPOSITION
0781 017D               M          DATA    do_setposition
                    02365          CMD_DEF do_reset,DO_RESET
                        M
0782 005A               M          DATA    DO_RESET
0783 0190               M          DATA    do_reset
                    02366          CMD_DEF do_stop,DO_STOP
                        M
0784 0073               M          DATA    DO_STOP
0785 0193               M          DATA    do_stop
                    02367          CMD_DEF do_capture,DO_CAPTURE
                        M
0786 0063               M          DATA    DO_CAPTURE
0787 0196               M          DATA    do_capture
                    02368          CMD_END
                        M ;
0788 0000               M          DATA    0x00
                    02369
                    02370
0789 0003           02371 PAR_TABLE        DATA    0x0003
078A 0020           02372                  DATA    VL
```

© 1997 Microchip Technology Inc.

```
078B 0103           02373                   DATA    0x0103
078C 0023           02374                   DATA    AL
078D 0202           02375                   DATA    0x0202
078E 0026           02376                   DATA    KP
078F 0302           02377                   DATA    0x0302
0790 0028           02378                   DATA    KV
0791 0402           02379                   DATA    0x0402
0792 002A           02380                   DATA    KI
0793 0501           02381                   DATA    0x0501
0794 002C           02382                   DATA    IM
0795 0602           02383                   DATA    0x0602
0796 002D           02384                   DATA    FV
0797 0702           02385                   DATA    0x0702
0798 002F           02386                   DATA    FA
0799 0008           02387                   DATA    NUMPAR
                    02388
                    02389           #if     DECIO
                    02390
079A 423F           02391 DEC_TABLE         DATA    0x423F
079B 000F           02392                   DATA    0x000F
079C 869F           02393                   DATA    0x869F
079D 0001           02394                   DATA    0x0001
079E 270F           02395                   DATA    0x270F
079F 0000           02396                   DATA    0x0000
07A0 03E7           02397                   DATA    0x03E7
07A1 0000           02398                   DATA    0x0000
07A2 0063           02399                   DATA    0x0063
07A3 0000           02400                   DATA    0x0000
07A4 0009           02401                   DATA    0x0009
07A5 0000           02402                   DATA    0x0000
07A6 FFFF           02403                   DATA    0xFFFF
                    02404           #endif
                    02405
                    02406
                    02407
                    02408
                    02409           END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X--------------- ---------------- XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
```


© 1997 Microchip Technology Inc.

```
0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0280 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
02C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0300 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0340 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0380 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
03C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0400 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0440 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0480 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
04C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0500 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0540 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0580 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
05C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0600 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0640 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0680 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
06C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0700 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0740 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0780 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXX--------- ----------------

All other memory blocks unused.

Program Memory Words Used:  1928

Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Saving and Restoring Status on Interrupt (Implementing a Parameter Stack)

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

The PIC17C42 has a 16 level deep hardware stack. The program counter is pushed into this stack on interrupts and subroutine calls. However, other key registers are not saved to the stack. Registers such as WREG, ALUSTA (which has carry, zero and other flag bits) and the bank select register (BSR) must be saved in an interrupt service routine. The following macros, PUSH and POP implement a parameter stack in data memory to save these register values.

The indirect addressing register, FSR0, is used to implement this parameter stack. It is assumed that FSR0 and its control bits are not used or modified elsewhere. The stack pointer (FSR0) is initialized at the highest RAM location (FFh).


```
Main_prog           SETF    FSR0            ;Initialize and dedicate FSR0 as stack pointer
                    BCF     ALUSTA,FS0      ;
                    BCF     ALUSTA,FS1      ;Set-up FSR0 for auto-dec
                    •
                    •
                    •
PUSH                MACRO
                    BCF     ALUSTA,FS0      ;Set-up FSR0 for auto-dec
                    MOVFP   ALUSTA,IND0     ;Save ALUSTA first
                    MOVFP   BSR,IND0        ;
                    MOVFP   W,IND0          ;
                    MOVFP   RAM_x, IND0     ;Now save general
                    MOVFP   RAM_y, IND0     ;Purpose registers
                    ENDM
POP                 MACRO
                    BSF     ALUSTA,FS0      ;Set-up for auto-inc
                    INCF    FSR0            ;
                    MOVPF   IND0, RAM_y     ;
                    MOVPF   IND0, RAM_x     ;
                    MOVPF   IND0,W          ;
                    MOVPF   IND0,BSR        ;
                    MOVFP   IND0,ALUSTA     ;restore ALUSTA last
                    DECF    FSR0            ;Adjust stack pointer
                    ENDM
                    •
                    •
                    •
interrupt_routine
                    PUSH                    ;save registers
                    •
                    •
                    •
;main body of interrupt service
                    •
                    •
                    •
                    POP                     ;restore status
                    RETFIE                  ;return
```

© 1997 Microchip Technology Inc.

While the macros are quite self-explanatory, the user should note a few subtle points.

- `MOVFP` instruction does not affect status flags while `MOVPF` does.
- `MOVFP` and `MOVPF` are used such that any register can be saved and restored. Note that the register being saved or restored has address 'f' (which can be anywhere from 00h to FFh) and the other address, IND0 (indirect), therefore, can be any address.
- FSR auto-increments or auto-decrements after the operation ('post'). Therefore, in the POP macro pre-increment is simulated.
- All interrupts should be disabled when executing the PUSH and POP macros. While PUSH will have the GIE bit disabled, POP may not necessarily have the GIE bit disabled. The user should disable the GIE bit when executing the POP.

Using this scheme, interrupts and subroutine calls can be nested, since the stack will grow and shrink. The stack can be used to pass parameters to subroutines.

© 1997 Microchip Technology Inc.

MICROCHIP

AN538

Using PWM to Generate Analog Output

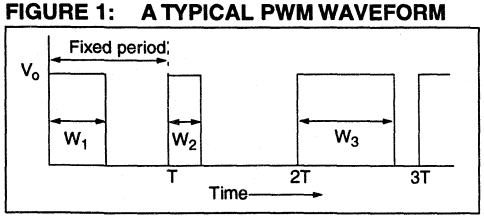
Author: *Amar Palacherla*
Microchip Technology Inc.

Pulse Width Modulation (PWM) modules, which produce basically digital waveforms, can be used as cheap Digital-to-Analog (D/A) converters only a few external components. A wide variety of microcontroller applications exist that need analog output but do not require high resolution D/A converters. Some speech applications (talk back units, speech synthesis systems in toys, etc.) also do not require high resolution D/A converters. For these applications, Pulse Width Modulated outputs may be converted to analog outputs.

Conversion of PWM waveforms to analog signals involves the use of analog low-pass filters. This application note describes the design criteria of the analog filters necessary and the requirements of the PWM frequency. Later in this application note, a simple RC low-pass filter is designed to convert PWM speech signals of 4 kHz bandwidth.

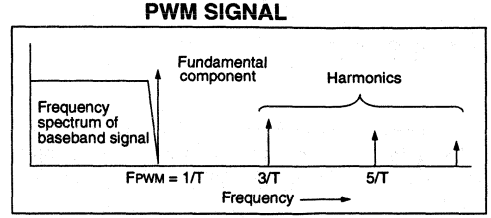
In a typical PWM signal, the base frequency is fixed, but the pulse width is a variable. The pulse width is directly proportional to the amplitude of the original unmodulated signal. In other words, in a PWM signal, the frequency of the waveform is a constant while the duty cycle varies (from 0% to 100%) according to the amplitude of the original signal. A typical PWM signal is shown in Figure 1.

FIGURE 1: A TYPICAL PWM WAVEFORM



A Fourier analysis of a typical PWM signal (such as the one depicted in Figure 1) shows that there is a strong peak at frequency $Fn = 1/T$. Other strong harmonics also exist at $F = K/T$, where K is an integer. These peaks are unwanted noise and should be eliminated. This requires that the PWM signal be low-pass filtered, thus eliminating these inherent noise components as shown in Figure 2.

FIGURE 2: FREQUENCY SPECTRUM OF A PWM SIGNAL

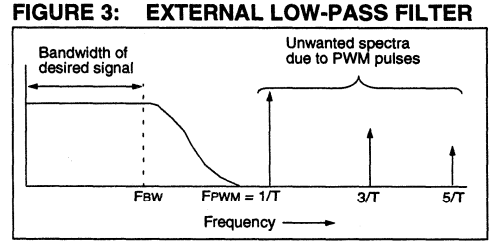


The band-width of the desired signal should be

$$F_{BW} <= (F_{PWM} = 1/T)$$

If FBW is selected such that FBW = FPWM, then the external low-pass filter should be a brick-wall type filter. Brick-wall type analog filters are very difficult and expensive to build. So, for practical purpose, the external low-pass filter should be as shown in Figure 3.

FIGURE 3: EXTERNAL LOW-PASS FILTER



This means, $F_{BW} << F_{PWM}$

$$or\ F_{PWM} >> F_{BW}$$

$$=> F_{PWM} = K \bullet F_{BW} \qquad (1)$$

where, K is a constant such that $K >> 1$

The value of K should be chosen dependant upon the number dB the inherent fundamental noise component of PWM will be rejected. An example follows:

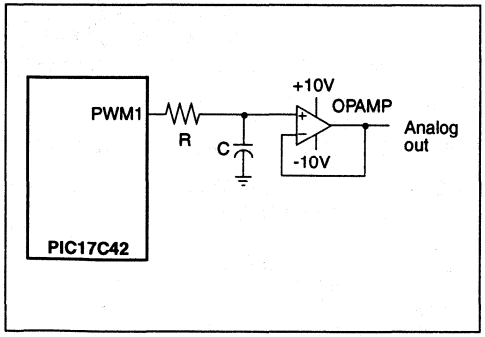
Example: It is required to design a simple RC low-pass filter to obtain an analog output from a pulse width modulated speech signal of bandwidth 4 kHz.

From *eqn (1)*, choosing arbitrarily K = 5,

$F_{PWM} = K \bullet f_{BW} = 5 \bullet 4\ kHz = 20\ kHz.$

© 1997 Microchip Technology Inc.

FIGURE 4: RC FILTER CONNECTED TO PWM1 OF PIC17C42



Choosing, the -3 dB point at 4 kHz, and using the relation RC = 1/(2 • π•f), we get R = 4 kΩ, if C is chosen as 0.01 µF:

R = 4.0 kΩ

C = 0.01 µF

Since the PWM frequency is selected as 20 kHz, the fundamental noise peak to be filtered is at 20 kHz. Now, lets calculate by how many dB the main peak of PWM signal is cut-off at 20 kHz:

$$(dB)\ 20\ kHz = -10 \bullet log[1 + (2\pi f.RC)2] = -14\ dB.$$

For many applications, this rejection of -14 dB will not suffice. Therefore instead of a simple RC low-pass filter, a higher order active low-pass filter may be necessary. Or, if the microcontroller is capable of modulating at higher PWM frequencies, the rejection of noise will be greater.

For example, using 8-bit resolution, the PIC17C42 can generate PWM frequency of 62.5 kHz.

At this frequency the attenuation of the PWM frequency is:

$$(dB)62.5\ kHz = -10 \bullet log[1 + (2\pi f.RC)2] = -24\ dB.$$

The higher frequency of the PIC17C42 PWM outputs makes it easier to generate analog output.

© 1997 Microchip Technology Inc.

MICROCHIP

AN539

Frequency and Resolution Options for PWM Outputs

Authors: *Stan D'Souza and Sumit Mitra*
Microchip Technology Inc.

INTRODUCTION

The PIC17C42 is equipped with two high frequency Pulse Width Modulation (PWM) outputs. In a pulse width modulated signal the period of the signal is (usually) kept fixed, while the duty cycle is varied. In this application note, we will discuss options in selecting PWM frequency and resolution.

This application note assumes that the internal clock is used for the time-base, which is typically the preferred setup. Also, throughout this application note, PWM1 output is used in examples, Timer1 is assumed to be the time-base.

Definition of terms:

Period of a PWM output is the duration after which the PWM pattern will repeat itself.

Frequency of a PWM output is = 1/Period.

Resolution of a PWM output is the granularity with which the duty cycle can be modulated.

In the case of the PIC17C42, when using PWM1 with Timer1 as the time-base:

PWM1 period = [(PR1) + 1] x 4TOSC

PWM1 duty cycle = (DC1) x TOSC

where PR1 = period register for Timer1

DC1 = PW1DCH, PW2DCL concantenated (10-bit value)

TOSC = oscillator period

At 16 MHz oscillator frequency, TOSC = 62.5 ns. The user can control the frequency of the PWM output by altering the 'period' value of the time-base. For example, if the period is chosen to be 100 TOSC (PR1 = 18h), then PWM frequency is:

1/(100 x 62.5) ns = 160 kHz.

Note however that the duty cycle resolution is a little less than 7-bits.

Useful and Common PWM Modes

While a variety of period values can be selected, the following modes would be most commonly used:

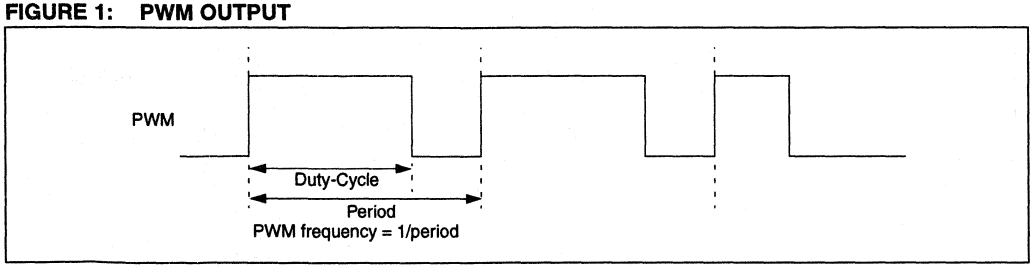
10-bit Mode: In this mode PWM duty cycle has full 10-bit resolution (maximum offered by the PIC17C42). The period register PR1 is set at FFh. PWM period = 1024TOSC = 64 µs. PWM frequency is 15.625 kHz. The user must write both PW1DCH and PW1DCL to update PWM output. See Appendix A for an example of code that modulates 10-bit resolution PWM output (PWM10.LST).

8-bit Hi-Resolution Mode: In this mode, the user has only an 8-bit quantity to write to the duty-cycle register. Period register is set at 3Fh (63 decimal), such that PWM period is 256Tosc. To write the 8-bit duty-cycle value, first the 8th bit is right shifted two bits. The upper six bits are written to PW1DCH and the lower two bits are written to PW1DCL as follows:

```
;8-bit duty-cycle value is in WREG
CLRF   TEMP          ;
RRCF   WREG          ;
RRCF   TEMP          ;
RRCF   WREG          ;
RRCF   TEMP          ;Shift right twice
ANDLW  b'00111111    ;Mask off two-high bits
MOVPF  WREG,PW1DCH ;Write duty-cycle values
MOVFP  TEMP,PW1DCL ;
```

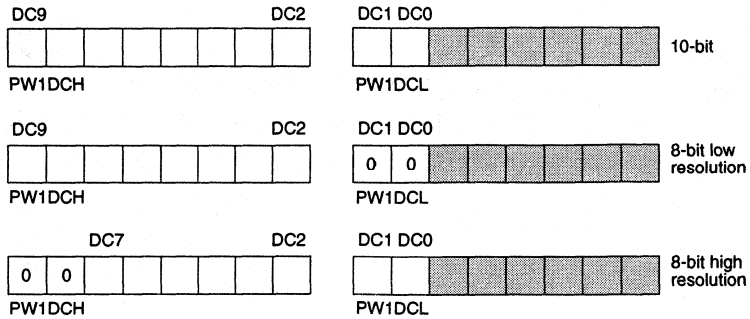
Note that in 8-bit, hi-resolution mode, maximum PWM frequency is attained. For example, at 16 MHz clock, PWM period = 256TOSC = 16 µs; PWM frequency = 62.5 kHz. See appendix B for an example code that generates 8-bit low high resolution PWM output (PWM8HI.LST).

FIGURE 1: PWM OUTPUT



© 1997 Microchip Technology Inc.

FIGURE 2: VARIOUS PWM MODES



Note: DC registers suited for 16-bit computation, using 10 MSb only.

8-bit Low Resolution Mode

In this mode, the user still has only an 8-bit quantity to write to the duty cycle register. However, the desired frequency of the PWM output is less, due to the nature of the application. For example, if the PWM output is being used to drive a motor through a power stage, the power transistors, (or devices) due to their switching time, will prefer PWM frequency not to exceed a certain frequency. In the previous section, we derived an 8-bit resolution PWM output at 62.5 kHz.

To attain a low-resolution PWM output, PW1DCL is always kept at zero. The 8-bit value is written to PW1DCH. The period (PR1) is set at FFh (i.e., 256 TCY equals 1024TOSC [15.625 kHz]). See Appendix C for an example code that produces 8-bit low resolution PWM output (`PWM8LO.LST`).

Choosing Resolution and Frequency of PWM Output

Actually, the resolution and the frequency of the PWM output is selectable within certain limits. The user will need to first define the requirements based on the application. There may be an upper limit to the frequency if the PWM is being used to drive motors. On the other hand, if the PWM is being filtered to generate an analog signal, higher frequency may be desirable. In any case, the lowest frequency achievable (using internal clock for the timer) is (FOSC/1024). At 16 MHz oscillator input, the lowest PWM frequency possible is 15.625 kHz. At resolutions less than 10-bit, higher frequencies are possible (Figure 3). For example, if 7-bit resolution is chosen, then the PWM frequencies can be 15.625 kHz, 31.25 kHz, 62.5 kHz or 125 kHz. The reader will note that it's how the 7-bits are placed within the 10-bit possible duty cycle value.

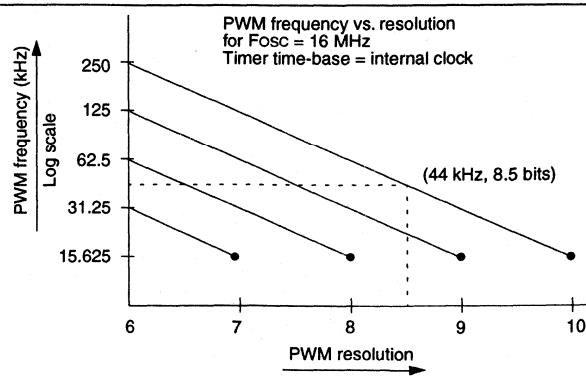
Conversely, if a certain frequency is desired, such as 44 kHz, then referring to Figure 1, resolution can be 8.5-bit or 7.5-bit or 6.5-bit etc.

Summary

The frequency and resolution of the PWM outputs of the PIC17C42 can be traded off against each other to best suit the application. The oscillator frequency can also be varied to adjust PWM frequency, if necessary. External clock should be used as timer time-base to generate very low frequency PWM output.

FIGURE 3: PWM FREQUENCY vs. RESOLUTION

At a given oscillator frequency (16 MHz in this figure), a family of curves represents the PWM freq/resolution combinations possible.



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: PWM10.LST

```
MPASM 01.40 Released           PWM8HI.ASM   1-16-1997  13:59:20         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         TITLE   "PULSE WIDTH MODULATION 8 BIT HIGH RESOLUTION"
                     00002         PROCESSOR      PIC17C42
                     00003
                     00004
                     00005 #include        "p17c42.inc"
                     00001         LIST
                     00002 ; P17C42.INC Standard Header File, Version 1.03 Microchip Technology, Inc.
                     00264         LIST
                     00006
  00000021           00007 PWM_HI  equ     0x21
  00000020           00008 PWM_LO  equ     0x20
  00000022           00009 TEMP    equ     0x22
                     00010 ;
  00000001           00011 F       equ     1
                     00012 ;
                     00013 ;The user would generate a 16 bit value which is saved in ram
                     00014 ;locations PWM_HI and PWM_LO byte. In 8 bit hi-res mode, the program
                     00015 ;transfers the 8 bit values to the lo Duty Cycle (DC) registers, to
                     00016 ;generate the required 8 bit hi-res PWM.
                     00028         LIST
                     00029 ;
                     00030 ;
                     00031 ;This is a short program to demonstrate how to generate PWM outputs with
                     00032 ;8 bit resolution. Since a 10Mhz crystal was used in the test system.
                     00033 ;The max. period = 256x100nS = 25.6uS or 39KHz. This program
                     00034 ;keeps the period constant and varies the duty cycle (which corresponds
                     00035 ;to the most significant 10 bits of the 16 bit value PWM_LO&PWM_HI).
                     00036 ;This program is interrupt driven, i.e. the update to the Duty cycle
                     00037 ;is done in the TMR0 interrupt, which then enables the pwm interrupt.
                     00038 ;The period update is done during the pwm interrupt. The pwm output
                     00039 ;ramps up from 0% to 100% duty cycle and then repeats. The full
                     00040 ;sweep takes approx. 13.3 secs.
                     00041 ;
                     00042 ;
                     00043 ;
                     00044 ;       Program:          PWM8HI.ASM
                     00045 ;       Revision Date:
                     00046 ;                         12-12-95      Compatibility with MPASMWIN 1.30
                     00047 ;
                     00048 ;*****************************************************************
                     00049 ;
0000                 00050         ORG     0
0000 C063            00051         goto    start
                     00052 ;
0010                 00053         ORG     0x10            ;vector for TMR0 interrupt
0010                 00054 TMR0_int
0010 C054            00055         goto    service_TMR0    ;service TMR0
                     00056 ;
0020                 00057         ORG     0x0020          ;vector for pwm interrupt
0020                 00058 pwm_int
0020 C046            00059         goto    service_pwm     ;service pwm only
                     00060 ;
0030                 00061         ORG     0x0030
                     00062 ;
```

```
                00063 ;initialize internal hardware to generate the pwm output
0030            00064 init_pwm8hi
0030 B802       00065         movlb   2
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0031 2910       00066         clrf    TMR1, F         ;clear timer 1
                00067                                 ;used to "drive" pwm1
0032 B062       00068         movlw   62              ;set period=39khz
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0033 0114       00069         movwf   PR1             ;       /
0034 B803       00070         movlb   3
0035 2922       00071         clrf    TEMP, F         ;TEMP = mask for pw1dcl
0036 6A21       00072         movfp   PWM_HI,WREG     ;get duty cyl. hi byte
0037 190A       00073         rrcf    WREG, F         ;rotate hi through carry
0038 1922       00074         rrcf    TEMP, F         ;rotate into lo byte
0039 190A       00075         rrcf    WREG, F         ;repeat for 2nd lsb
003A 1922       00076         rrcf    TEMP, F         ;       /
003B B53F       00077         andlw   B'00111111'     ;mask hi bits
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
003C 4012       00078         movpf   W,PW1DCH        ;save in high
Warning[202]: Argument out of range.  Least significant bits used.
003D 7022       00079         movfp   TEMP,PW1DCL     ;save in low
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
003E 2916       00080         clrf    TCON1, F        ;tmr1 inc. internally
                00081                                 ;as 8 bit counter
003F B011       00082         movlw   B'00010001'     ;start tmr1 and
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0040 4017       00083         movpf   W,TCON2         ;enable pwm1
0041 B801       00084         movlb   1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0042 2917       00085         clrf    PIE, F          ;clr all int. enables
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0043 2916       00086         clrf    PIR, F          ;clear all interrupts
0044 8307       00087         bsf     INTSTA,PEIE     ;except peripheral int.
0045 0005       00088         retfie
                00089 ;
                00090 ;
                00091 ;everytime a new value is written to the PWM_HI, PWM_LO regs, the
                00092 ;tmr1 interrupt is enabled. The DC value are written just before
                00093 ;the "pwm interrupt" is enabled. Here the new period register is
                00094 ;updated. In this example, period is kept constant at 62 Tcyl.
0046            00095 service_pwm
                00096         ;if the period changed, write new value here.
0046 B802       00097         movlb   2               ;select bank 2
0047 B062       00098         movlw   62              ;period = 62 Tcyl.
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0048 0114       00099         movwf   PR1             ;       /
0049 B801       00100         movlb   1               ;disable tmr1 int
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
004A 8C17       00101         bcf     PIE,TMR1IE      ;       /
004B 0005       00102         retfie
                00103 ;
                00104         PAGE
                00105 ;This part of the program is basically used to simmulate a change
                00106 ;which would be used to drive the pwm output.
                00107 ;
                00108 ;the TMR0 is set up to interrupt every 52 mS.
004C            00109 init_TMR0
004C B020       00110         movlw   B'00100000'     ;set up TMR0 timer
004D 650A       00111         movfp   WREG,T0STA      ;       /
004E 290B       00112         clrf    TMR0L, F        ;clear TMR0
004F 290C       00113         clrf    TMR0H, F        ;       /
0050 B031       00114         movlw   31              ;init pwm at 50%
0051 0121       00115         movwf   PWM_HI          ;save in high
0052 8107       00116         bsf     INTSTA,T0IE     ;enable TMR0 int.
0053 0002       00117         return
                00118 ;
```


© 1997 Microchip Technology Inc.

```
                00119 ;Every TMR0 interrupt, the PWM_HI&PWM_LO bytes are incremented by 1.
                00120 ;Only the 8 most significant bits are incremented.
                00121 ;
0054            00122 service_TMR0
0054 8D07       00123         bcf     INTSTA,T0IF     ;reset int flag
                00124         ;do a pseudo inc of the 8 bit PWM_HI.
0055 1521       00125         incf    PWM_HI, F       ;inc PWM_HI
                00126         ;now load the values into the Duty Cycle registers
0056 B803       00127         movlb   3               ;bank 3
0057 2922       00128         clrf    TEMP, F         ;TEMP = mask for pw1dcl
0058 6A21       00129         movfp   PWM_HI,WREG     ;get duty cyl. hi byte
0059 190A       00130         rrcf    WREG, F         ;rotate hi through carry
005A 1922       00131         rrcf    TEMP, F         ;rotate into lo byte
005B 190A       00132         rrcf    WREG, F         ;repeat for 2nd lsb
005C 1922       00133         rrcf    TEMP, F         ;       /
005D B53F       00134         andlw   B'00111111'     ;mask hi bits
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
005E 4A12       00135         movpf   WREG,PW1DCH     ;save in high
Warning[202]: Argument out of range.  Least significant bits used.
005F 7022       00136         movfp   TEMP,PW1DCL     ;save in low
0060 B801       00137         movlb   1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0061 8417       00138         bsf     PIE,TMR1IE      ;enable tmr1 int
0062 0005       00139         retfie
                00140 ;
                00141         PAGE
                00142 ;
0063            00143 start
0063 8406       00144         bsf     CPUSTA,GLINTD   ;disable interrupts
0064 E04C       00145         call    init_TMR0       ;initailize the TMR0 tmr
                00146                                 ;for test purposes
0065 E030       00147         call    init_pwm8hi     ;initialize 8 bit pwm
0066 C066       00148 loop    goto    loop            ;spin wheels.
                00149 ;
                00150
                00151         END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X--------------- X--------------- X--------------- XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXX--------- ----------------

All other memory blocks unused.

Program Memory Words Used:    58

Errors   :     0
Warnings :     2 reported,     0 suppressed
Messages :    11 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: PWM8HI.LST

```
MPASM 01.40 Released           PWM8HI.ASM   1-16-1997  13:59:20         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         TITLE   "PULSE WIDTH MODULATION 8 BIT HIGH RESOLUTION"
                     00002         PROCESSOR       PIC17C42
                     00003
                     00004
                     00005 #include        "p17c42.inc"
                     00001         LIST
                     00002 ;P17C42.INC Standard Header File, Version 1.03 Microchip Technology, Inc.
                     00264         LIST
                     00006
  00000021           00007 PWM_HI  equ     0x21
  00000020           00008 PWM_LO  equ     0x20
  00000022           00009 TEMP    equ     0x22
                     00010 ;
  00000001           00011 F       equ     1
                     00012 ;
                     00013 ;The user would generate a 16 bit value which is saved in ram
                     00014 ;locations PWM_HI and PWM_LO byte. In 8 bit hi-res mode, the program
                     00015 ;transfers the 8 bit values to the lo Duty Cycle (DC) registers, to
                     00016 ;generate the required 8 bit hi-res PWM.
                     00028         LIST
                     00029 ;
                     00030 ;
                     00031 ;This is a short program to demonstrate how to generate PWM outputs with
                     00032 ;8 bit resolution. Since a 10Mhz crystal was used in the test system.
                     00033 ;The max. period = 256x100nS = 25.6uS or 39KHz. This program
                     00034 ;keeps the period constant and varies the duty cycle (which corresponds
                     00035 ;to the most significant 10 bits of the 16 bit value PWM_LO&PWM_HI).
                     00036 ;This program is interrupt driven, i.e. the update to the Duty cycle
                     00037 ;is done in the TMR0 interrupt, which then enables the pwm interrupt.
                     00038 ;The period update is done during the pwm interrupt. The pwm output
                     00039 ;ramps up from 0% to 100% duty cycle and then repeats. The full
                     00040 ;sweep takes approx. 13.3 secs.
                     00041 ;
                     00042 ;
                     00043 ;
                     00044 ;       Program:          PWM8HI.ASM
                     00045 ;       Revision Date:
                     00046 ;                         12-12-95      Compatibility with MPASMWIN 1.30
                     00047 ;
                     00048 ;*****************************************************************
                     00049 ;
0000                 00050         ORG  0
0000 C063            00051         goto    start
                     00052 ;
0010                 00053         ORG     0x10            ;vector for TMR0 interrupt
0010                 00054 TMR0_int
0010 C054            00055         goto    service_TMR0    ;service TMR0
                     00056 ;
0020                 00057         ORG     0x0020          ;vector for pwm interrupt
0020                 00058 pwm_int
0020 C046            00059         goto    service_pwm     ;service pwm only
                     00060 ;
0030                 00061         ORG     0x0030
                     00062 ;
```


© 1997 Microchip Technology Inc.

```
                00063 ;initialize internal hardware to generate the pwm output
0030            00064 init_pwm8hi
0030 B802       00065         movlb   2
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0031 2910       00066         clrf    TMR1, F         ;clear timer 1
                00067                                 ;used to "drive" pwm1
0032 B062       00068         movlw   62              ;set period=39khz
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0033 0114       00069         movwf   PR1             ;       /
0034 B803       00070         movlb   3
0035 2922       00071         clrf    TEMP, F         ;TEMP = mask for pw1dcl
0036 6A21       00072         movfp   PWM_HI,WREG     ;get duty cyl. hi byte
0037 190A       00073         rrcf    WREG, F         ;rotate hi through carry
0038 1922       00074         rrcf    TEMP, F         ;rotate into lo byte
0039 190A       00075         rrcf    WREG, F         ;repeat for 2nd lsb
003A 1922       00076         rrcf    TEMP, F         ;       /
003B B53F       00077         andlw   B'00111111'     ;mask hi bits
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
003C 4012       00078         movpf   W,PW1DCH        ;save in high
Warning[202]: Argument out of range.  Least significant bits used.
003D 7022       00079         movfp   TEMP,PW1DCL     ;save in low
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
003E 2916       00080         clrf    TCON1, F        ;tmr1 inc. internally
                00081                                 ;as 8 bit counter
003F B011       00082         movlw   B'00010001'     ;start tmr1 and
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0040 4017       00083         movpf   W,TCON2         ;enable pwm1
0041 B801       00084         movlb   1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0042 2917       00085         clrf    PIE, F          ;clr all int. enables
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0043 2916       00086         clrf    PIR, F          ;clear all interrupts
0044 8307       00087         bsf     INTSTA,PEIE     ;except peripheral int.
0045 0005       00088         retfie
                00089 ;
                00090 ;
                00091 ;everytime a new value is written to the PWM_HI, PWM_LO regs, the
                00092 ;tmr1 interrupts is enabled. The DC value are written just before
                00093 ;the "pwm interrupt" is enabled. Here the new period register is
                00094 ;updated. In this example, period is kept constant at 62 Tcyl.
0046            00095 service_pwm
                00096         ;if the period changed, write new value here.
0046 B802       00097         movlb   2               ;select bank 2
0047 B062       00098         movlw   62              ;period = 62 Tcyl.
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0048 0114       00099         movwf   PR1             ;       /
0049 B801       00100         movlb   1               ;disable tmr1 int
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
004A 8C17       00101         bcf     PIE,TMR1IE      ;       /
004B 0005       00102         retfie
                00103 ;
                00104         PAGE
                00105 ;This part of the program is basically used to simmulate a change
                00106 ;which would be used to drive the pwm output.
                00107 ;
                00108 ;the TMR0 is set up to interrupt every 52 mS.
004C            00109 init_TMR0
004C B020       00110         movlw   B'00100000'     ;set up TMR0 timer
004D 650A       00111         movfp   WREG,T0STA      ;       /
004E 290B       00112         clrf    TMR0L, F        ;clear TMR0
004F 290C       00113         clrf    TMR0H, F        ;       /
0050 B031       00114         movlw   31              ;init pwm at 50%
0051 0121       00115         movwf   PWM_HI          ;save in high
0052 8107       00116         bsf     INTSTA,T0IE     ;enable TMR0 int.
0053 0002       00117         return
                00118 ;
```


© 1997 Microchip Technology Inc.

```
                00119 ;Every TMR0 interrupt, the PWM_HI&PWM_LO bytes are incremented by 1.
                00120 ;Only the 8 most significant bits are incremented.
                00121 ;
0054            00122 service_TMR0
0054 8D07       00123         bcf     INTSTA,T0IF     ;reset int flag
                00124         ;do a pseudo inc of the 8 bit PWM_HI.
0055 1521       00125         incf    PWM_HI, F       ;inc PWM_HI
                00126         ;now load the values into the Duty Cycle registers
0056 B803       00127         movlb   3               ;bank 3
0057 2922       00128         clrf    TEMP, F         ;TEMP = mask for pw1dc1
0058 6A21       00129         movfp   PWM_HI,WREG     ;get duty cyl. hi byte
0059 190A       00130         rrcf    WREG, F         ;rotate hi through carry
005A 1922       00131         rrcf    TEMP, F         ;rotate into lo byte
005B 190A       00132         rrcf    WREG, F         ;repeat for 2nd lsb
005C 1922       00133         rrcf    TEMP, F         ;       /
005D B53F       00134         andlw   B'00111111'     ;mask hi bits
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
005E 4A12       00135         movpf   WREG,PW1DCH     ;save in high
Warning[202]: Argument out of range.  Least significant bits used.
005F 7022       00136         movfp   TEMP,PW1DCL     ;save in low
0060 B801       00137         movlb   1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0061 8417       00138         bsf     PIE,TMR1IE          ;enable tmr1 int
0062 0005       00139         retfie
                00140 ;
                00141         PAGE
                00142 ;
0063            00143 start
0063 8406       00144         bsf     CPUSTA,GLINTD         ;disable interrupts
0064 E04C       00145         call    init_TMR0       ;initailize the TMR0 tmr
                00146                                 ;for test purposes
0065 E030       00147         call    init_pwm8hi     ;initialize 8 bit pwm
0066 C066       00148 loop    goto    loop            ;spin wheels.
                00149 ;
                00150
                00151         END
PULSE WIDTH MODULATION 8 BIT HIGH RESOLUTION
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X--------------- X--------------- X--------------- XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXX--------- ----------------

All other memory blocks unused.

Program Memory Words Used:    58


Errors   :     0
Warnings :     2 reported,     0 suppressed
Messages :    11 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C: PWM8LO.LST

```
MPASM 01.40 Released           PWM8LO.ASM   1-16-1997  13:41:45         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         TITLE   "PULSE WIDTH MODULATION 8 BIT LOW RESOLUTION"
                     00002         PROCESSOR      PIC17C42
                     00003
                     00004
                     00005 #include        "p17c42.inc"
                     00001         LIST
                     00002 ; P17C42.INC Standard Header File, Version 1.03 Microchip Technology, Inc.
                     00264         LIST
                     00006
  00000021           00007 PWM_HI  equ     0x21
  00000020           00008 PWM_LO  equ     0x20
  00000022           00009 TEMP    equ     0x22
                     00010 ;
  00000001           00011 F       EQU     1
                     00012 ;
                     00013 ;The user would generate a 16 bit value which is saved in ram
                     00014 ;locations PWM_HI and PWM_LO byte. In 8 bit lo-res mode, the program
                     00015 ;transfers the 8 hi-byte value directly to the PW1DCH register.
                     00027         LIST
                     00028 ;
                     00029 ;
                     00030 ;This is a short program to demonstrate how to generate PWM outputs with
                     00031 ;8 bit low resolution. Since a 5.068Mhz crystal was used in the test system.
                     00032 ;The max. period = 1024x100nS = 102.4 uS or 9.8 KHz. This program
                     00033 ;keeps the period constant and varies the duty cycle (which corresponds
                     00034 ;to the most significant 8 bits of the 16 bit value PWM_LO&PWM_HI).
                     00035 ;This program is interrupt driven, i.e. the update to the Duty cycle
                     00036 ;is done in the TMR0 interrupt, which then enables the pwm interrupt.
                     00037 ;The period update is done during the pwm interrupt. The pwm output
                     00038 ;ramps up from 0% to 100% duty cycle and then repeats. The full
                     00039 ;sweep takes approx. 52 secs.
                     00040 ;
                     00041 ;
                     00042 ;
                     00043 ;       Program:          PWM8LO.ASM
                     00044 ;       Revision Date:
                     00045 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00046 ;
                     00047 ;*****************************************************************
                     00048 ;
0000                 00049         ORG     0
0000 C053            00050         goto    start
                     00051 ;
0010                 00052         ORG     0x10            ;vector for TMR0 interrupt
0010                 00053 TMR0_int
0010 C04C            00054         goto    service_TMR0    ;service TMR0
                     00055 ;
0020                 00056         ORG     0x0020          ;vector for pwm interrupt
0020                 00057 pwm_int
0020 C03E            00058         goto    service_pwm     ;service pwm only
                     00059 ;
0030                 00060         ORG     0x0030
                     00061 ;
                     00062 ;initialize internal hardware to generate the output
```

```
                00063 ;for 8 bit low resolution pwm
0030            00064 init_pwm8lo
0030 B802       00065         movlb   2
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0031 2910       00066         clrf    TMR1, F         ;clear timer 1
                00067                                 ;used to "drive" pwm1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0032 2B14       00068         setf    PR1, F          ;set period=9.8 khz
0033 B803       00069         movlb   3
Warning[202]: Argument out of range.  Least significant bits used.
0034 7221       00070         movfp   PWM_HI,PW1DCH   ;load duty cyl. hi byte
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0035 2910       00071         clrf    PW1DCL, F       ;clear lo byte
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0036 2916       00072         clrf    TCON1, F        ;TMR1 inc. internally
                00073                                 ;as 8 bit counter
0037 B011       00074         movlw   B'00010001'     ;start TMR1 and
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0038 4A17       00075         movpf   WREG,TCON2      ;enable pwm1
0039 B801       00076         movlb   1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
003A 2917       00077         clrf    PIE, F          ;clr all int. enables
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
003B 2916       00078         clrf    PIR, F          ;clear all interrupts
003C 8307       00079         bsf     INTSTA,PEIE     ;except peripheral int.
003D 0005       00080         retfie
                00081 ;
                00082 ;
                00083 ;everytime a new value is written to the PWM_HI, PWM_LO regs, the
                00084 ;TMR1 interrupts is enabled. The DC value are written just before
                00085 ;the "pwm interrupt" is enabled. Here the new period register is
                00086 ;updated. In this example, period is kept constant at 0xff Tcyl.
003E            00087 service_pwm
                00088         ;if the period changed, write new value here.
003E B802       00089         movlb   2               ;select bank 2
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
003F 2B14       00090         setf    PR1, F          ;period <-- 0xff
0040 B801       00091         movlb   1               ;disable TMR1 int
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0041 8C17       00092         bcf     PIE,TMR1IE      ;       /
0042 0005       00093         retfie
                00094 ;
                00095         PAGE
                00096 ;This part of the program is basically used to simmulate a change
                00097 ;which would be used to drive the pwm output.
                00098 ;
                00099 ;the TMR0 is set up to interrupt every 52 mS.
0043            00100 init_TMR0
0043 B020       00101         movlw   B'00100000'     ;set up TMR0 timer
0044 6500       00102         movfp   W,T0STA         ;       /
0045 290B       00103         clrf    TMR0L, F        ;clear TMR0
0046 290C       00104         clrf    TMR0H, F        ;       /
0047 B080       00105         movlw   0x80            ;begin PWM at 50% dc
0048 0121       00106         movwf   PWM_HI          ;       /
0049 2920       00107         clrf    PWM_LO, F       ;       /
004A 8107       00108         bsf     INTSTA,T0IE     ;enable TMR0 int.
004B 0002       00109         return
                00110 ;
                00111 ;Every TMR0 interrupt, the PWM_HI&PWM_LO bytes are incremented by 1.
                00112 ;Only the 8 most significant bits are incremented.
                00113 ;
004C            00114 service_TMR0
004C 8D07       00115         bcf     INTSTA,T0IF           ;reset int flag
                00116         ;do a inc of the 8 bit PWM_HI
004D 1521       00117         incf    PWM_HI, F
                00118         ;now load the values into the Duty Cycle registers
```


© 1997 Microchip Technology Inc.

```
004E B803      00119          movlb   3
Warning[202]: Argument out of range.  Least significant bits used.
004F 7221      00120          movfp   PWM_HI,PW1DCH   ;load hi byte
0050 B801      00121          movlb   1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0051 8417      00122          bsf     PIE,TMR1IE      ;enable TMR1 int
0052 0005      00123          retfie
               00124 ;
               00125          PAGE
               00126 ;
0053           00127 start
0053 8406      00128          bsf     CPUSTA,GLINTD   ;disable interrupts
0054 E043      00129          call    init_TMR0       ;initailize the TMR0 Timer
               00130                                  ;for test purposes
0055 E030      00131          call    init_pwm8lo     ;initialize pwm
0056 C056      00132 loop     goto    loop            ;spin wheels.
               00133 ;
               00134
               00135          END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X--------------- X--------------- X--------------- XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXX--------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:    42


Errors   :     0
Warnings :     2 reported,     0 suppressed
Messages :    10 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN540

Implementing IIR Digital Filters

Author: *Amar Palacherla*
Microchip Technology Inc.

INTRODUCTION

This application note describes the implementation of various digital filters using the PIC17C42, the first member of Microchip's 2nd generation of 8-bit microcontrollers. The PIC17C42 is a very high speed 8-bit microcontroller with an instruction cycle time of 250 ns (@ 16 MHz input clock). Even though the PIC17C42 is an 8-bit device, it's high speed and efficient instruction set allows implementation of digital filters for practical applications. Traditionally digital filters have been implemented using expensive Digital Signal Processors (DSPs). In a system the DSP is normally a slave processor being controlled by either an 8-bit or 16-bit microcontroller. Where sampling rates are not high (especially in mechanical control systems), a single chip solution is possible using the PIC17C42.

This application note provides a few examples of implementing digital filters. Example code for 2nd order Infinite Impulse Response (IIR) filters is given. The following type of filters are implemented:

- Low Pass
- High Pass
- Band Pass
- Band Stop (notch) filter

This application note does not explain how to design a filter. Filter design theory is well established and is beyond the scope of this application note. It is assumed that a filter is designed according to the desired specifications. The desired digital filters may be designed using either standard techniques or using commonly available digital filter design software packages.

Finite Impulse Response (FIR) filters have many advantages over IIR filters, but are much more resource intensive (both in terms of execution time and RAM). On the other hand, IIR filters are quite attractive for implementing with the PIC17C42 resources. Especially where phase information is not so important, IIR filters are a good choice (FIR filters have a linear phase response). Of the various forms used for realizing digital filters (like, Direct form, Direct II form, Cascade form, Parallel, Lattice structure, etc.) the Direct II form is used in this application note. It is easy to understand and simple macros can be built using these structures.

THEORY OF OPERATION

Digital filters in most cases assume the following form of relationship between the output and input sequences.

$$y(n) = \sum_{i=o}^{M} a_i y(n-i) + \sum_{j=o}^{N} b_j x(n-j)$$

The above equation basically states that the present output is a weighted sum of the past inputs and past outputs. In case of FIR filters, the weighted constants $ai = 0$ and in case of IIR filters, at least one of the ai constants is non zero. In case of IIR, the above formula may be rewritten in terms of Z transform as:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=o}^{M} b_k Z^{-k}}{1 + \sum_{k=1}^{N} a_k Z^{-k}}$$

The above equation can further be rewritten in difference equation format as follows:

$$y(n) = \sum_{i=1}^{M} a_i y(n-i) + \sum_{j=o}^{N} b_j x(n-j)$$

© 1997 Microchip Technology Inc.

Realization of the above equation is called the Direct Form II structure. For example, in case of a second order structure, $M = N = 2$, gives the following difference equations:

EQUATION 1:

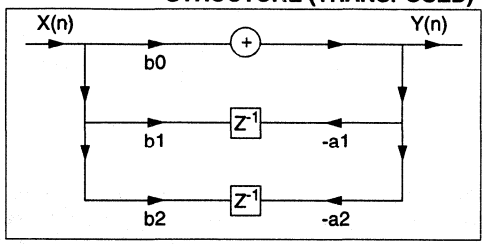
$$d(n) = x(n) + a_1 d(n-1) + a_2 d(n-2)$$

EQUATION 2:

$$y(n) = b_0 d(n) + b_1 d(n-1) + b_2 (d(n-2))$$

The above difference equations may be represented as shown in Figure 1.

FIGURE 1: 2ND ORDER DIRECT FORM II STRUCTURE (TRANSPOSED)



The structure as shown in Figure 1 may be cascaded to attain a higher order filter. For example, if two stages are cascaded together, a 4th Order IIR Filter is obtained. This way, the output of the 1st stage becomes the input to the second stage. Multiple order filters are thus implemented by cascading a 2nd order filter structure as shown in Figure 1.

IMPLEMENTATION

A 4th order IIR Filter is implemented by cascading two of the structures shown in Figure 1. The output Y (output of each filter stage) is computed by direct implementation of Equation 1 and Equation 2. Since each stage is similar algorithmically, it is implemented as a macro using Microchip's, Assembler/Linker for PIC17C42. This Macro (labelled `BIQUAD`) is called twice for implementing a 4th order filter. The output of the 1st stage is directly fed to the input of the second stage without any scaling.

Scaling may be required depending on the particular application. The user can modify the code very easily without any penalty on speed. Also, saturation arithmetic is not used. Overflows can be avoided by limiting the input sequence amplitude. All numbers are assumed to be 16 bits in Q15 format (15 decimal points, MSb is sign bit). Thus the user must scale and sign extend the input sequence accordingly. For example, if the input is from a 12-bit A/D converter, the user must sign extend the 12-bit input if bit 11 is a one.

The `BIQUAD` macro is a generic macro and can be used for all IIR filters whether it is Low Pass, High Pass, Band Pass or Band Stop. A general purpose 16x16 multiplier routine is also provided. This routine is implemented as a straight line code for speed considerations.

The 4th order IIR filter implemented is a Low Pass Filter with the specifications shown in Table 1.

TABLE 1: FILTER CONSTANTS

	BAND1	BAND2
Lower Band Edge	0.0	600 Hz
Upper Band Edge	500 Hz	1 kHz
Nominal Gain	1.0	0.0
Nominal Ripple	0.01	0.05
Maximum Ripple	0.00906	0.04601
Ripple in dB	0.07830	-26.75
Sampling Frequency = 2 kHz		

The Low Pass Filter is designed using a digital filter design package (DFDP™ by Atlanta Signal Processors Inc.). The filter package produces filter constants of the structure shown in Table 1. Table 2 shows the filter co-efficients that are obtained for the above Low Pass filter specification.

TABLE 2: FILTER CO-EFFICIENTS

Stage	Co-efficients				
	a1	a2	b0	b1	b2
1	-0.133331	0.167145	0.285431	0.462921	0.285431
2	0.147827	0.765900	0.698273	0.499908	0.698273

The above filter co-efficients (5 per stage) are quantized to Q15 format (i.e they are multiplied by 32768) and saved in program memory (starting at label `_coeff_lpass`). The constants for both the stages are read into data memory using `TLRD` and `TABLRD` instructions in the Initialization Routine (labelled `initFilter`). The user may read the coefficients of only one stage at a time and save some RAM at the expense of speed.

The sample 4th order Low Pass IIR Filter is tested by analyzing the impulse response of the filter. An impulse signal is fed as input to the filter. This is simulated by forcing the input to the filter by a large quantity (say 7F00h) on the first input sample, and the all zeros from the 2nd sample onwards. The output sequence is the filter's impulse response and is captured into the PICMASTER's (Microchip's Universal In-Circuit Emulator) real-time trace buffer. This captured data from PICMASTER is saved to file and analyzed. Analysis was done using MathCad™ for Windows® and a DSP Analysis program from Burr-Brown (DSPLAY™). The Fourier Transform of this impulse response of the filter should display the filter's frequency response, in this case being a Low Pass type. The plots of the impulse response and the frequency response are shown in Figure 2, Figure 3 and Figure 4.

MathCad is a trademark of MathSoft, Inc.
DSPLAY is a trademark of Burr-Brown
DFDP is a trademark of Atlanta Signal Processing Inc.
Windows is a registered trademark of Microsoft Corporation.

© 1997 Microchip Technology Inc.

FIGURE 2: IMPULSE RESPONSE CAPTURED FROM PICMASTER

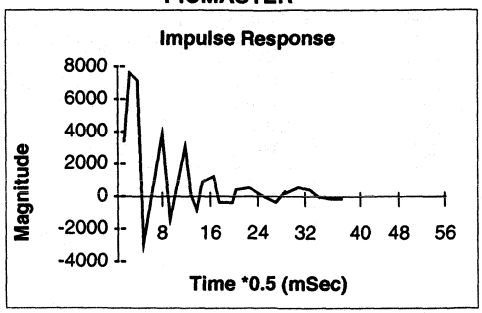
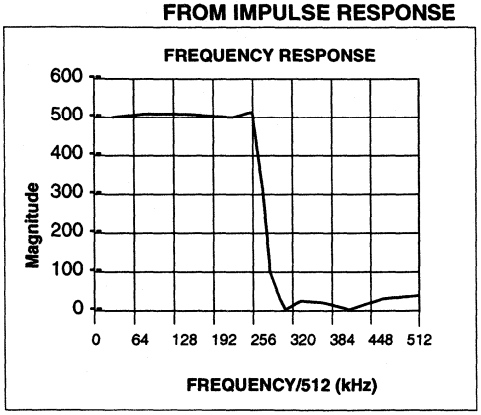


FIGURE 3: SPECTRUM COMPUTED FROM IMPULSE RESPONSE



PERFORMANCE

The resource requirements for filter implementations using a PIC17C42 is given in Table 3. These numbers can be used to determine whether a higher order filter can be executed in real-time. The same information may be used to determine the highest sampling rate possible.

FILTER APPLICATIONS

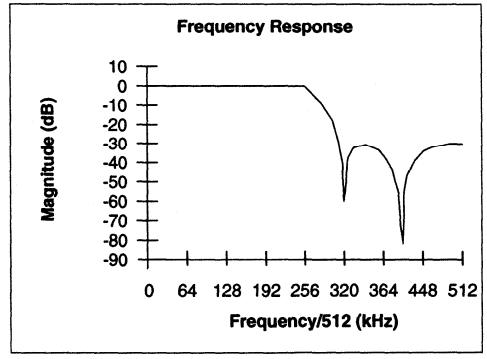
Digital filters find applications in many areas especially those involving processing of real world signals. In some applications like ABS systems in an automobile, digital filtering becomes a must. In this case elimination of noise (especially glitches and false readings of sensors) is very critical and thus becomes a requirement of digital signal processing.

TABLE 3: RESOURCE REQUIREMENTS

Timing (Cycles)(1)	#of Filter Stages*775 + 16
Program Memory (locations)(1)	#of Filter Stages*68 + 290
RAM (File Registers)	#of Filter Stages*16 + 16

Note 1: The above numbers do not include the initialization routine.

FIGURE 4: LOG MAGNITUDE SPECTRUM OF IMPULSE RESPONSE



Digital filters are also needed in Process Control where notch filters and low pass filters are desired because the signals from sensors are transmitted over long lines, especially in a very noisy environment. In these cases, typically a notch filter (centering 50 Hz or 60 Hz) is used. In cases of eliminating background noise, a band stop filter (e.g., 40 Hz to 120 Hz) is used. The sample code given in this application note can be used to design a feedback control system's digital compensator. For example, a typical DC Motor's digital compensator (like a dead-beat compensator) is of second order and has the same filter structure that is implemented in this application note.

© 1997 Microchip Technology Inc.

TABLE 4: RESOURCE REQUIREMENTS

Filter Order	Cycles	Real Time (@16 MHz)	Maximum Sampling	Program Memory(1)	RAM
2	791	197.75 µs	5.05 kHz	358	32
4	1566	391.5 µs	2.55 kHz	426	48
6	2341	585.25 µs	1.7 kHz	494	64
8	3116	779.0 µs	1.28 kHz	562	80
10	3891	972.75 µs	1.0 kHz	630	94

Note 1: The above numbers do not include the initialization routine.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: IIR.LS0T

```
MPASM 01.40 Released              IIR.ASM   1-16-1997  14:48:37         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         TITLE   "Digital IIR Filter Using PIC17C42"
                     00002
                     00003         LIST    P=17C42, columns=120,WRAP, R = DEC
                     00004
                     00005                 #include <p17c42.inc>
                     00001         LIST
                     00002 ;P17C42.INC Standard Header File, Version 1.03 Microchip Technology, Inc.
                     00264         LIST
                     00006
                     00007         #define true    1
                     00008         #define false   0
                     00009         #define TRUE    1
                     00010         #define FALSE   0
                     00011
                     00012         #define LSB     0
                     00013         #define MSB     7
                     00014
00000001             00015 _INC     equ     1
00000000             00016 _NO_INC  equ     0
00000000             00017 _LOW     equ     0
00000001             00018 _HIGH    equ     1
                     00019
                     00020                 #include <17c42.mac>
                     00945         LIST
                     00021
                     00022                 #include <17c42iir.mac>
                     00001         LIST
                     00002 ;*****************************************************************
                     00003 ;                       PIC17C42 MACRO
                     00004 ;
                     00005 ;Macro For A Bi-Quad IIR Filter
                     00006 ;       2nd order Direct Form (Transposed) Type
                     00007 ;
                     00008 ; Filter co-efficients B0 & B2 are assumed equal
                     00009 ;
                     00010 ; The difference equations for each cascade section is given by :
                     00011 ;         Y(n) = B0*D(n) + B1*D(n-1) + B2*D(n-2)
                     00012 ;         D(n) = X(n) - A1*D(n-1) -  A2*D(n-2)
                     00013 ;    where X(n) = input sample, Y(n) = output of filter
                     00014 ;    and A1, A2, B0, B1, B2 are the Filter Co-efficients
                     00015 ;
                     00016 ;  The above difference equations are only for 1 section of a
                     00017 ; 2nd order Direct_Form II Filter structure (IIR)
                     00018 ;
                     00019 ;   NOTE :
                     00020 ;       It is possible to design the above structures
                     00021 ;       such that the co-efficients B0 = B2. If this is the
                     00022 ;       case,
                     00023 ;       Y(n) = B0*[D(n) + D(n-2)] = B2*[D(n) + D(n-2)]
                     00024 ;       This way, one multiplication can be avoided
                     00025 ;
                     00026 ;  If a 4th order filter is to be implemented, the output of
                     00027 ; the 1st structure should be input to the 2nd cascade section
                     00028 ;
```


© 1997 Microchip Technology Inc.

```
                00029 ;       Timing (WORST CASE) :
                00030 ;              59+4*179 = 775 Cycles
                00031 ;              (194 uS @ 16 Mhz)
                00032 ;              Program Memory  :
                00033 ;              63 locations
                00034 ;
                00035 ;*****************************************************************
                00036 ;       The sample filters are designed so that B0=B2
                00037 ;       This saves 1 multiplication
                00038 ;
00000001        00039 B0_EQUALS_B2    equ     TRUE
                00040 ;
                00041 ;*****************************************************************
                00042 ;  Parameters to BIQUAD Macro :
                00043 ;               Filter Constants A1, A2, B0, B1, B2
                00044 ;               & D(n), D(n-1), D(n-2), filter stage #
                00045 ;
                00046 BIQUAD  MACRO   Ax1,Ax2,Bx0,Bx1,Dn,Dn_1,Dn_2,stage
                00047
                00048 ;
                00049 ; Compute Ax2*D(n-2)
                00050 ;
                00051  MOVFP16  Dn_2,AARG      ; D(n-2) = multiplier
                00052  MOVFP16  Ax2,BARG       ; A2 = multiplicand
                00053         call    DblMult  ; (ACCd,ACCc) = A2*D(n-2)
                00054 ;
                00055 ; Add product to output of 1st section
                00056 ; Save result in 32 bit Accumulator
                00057 ;
                00058         ADD32    DPX,ACC
                00059 ;
                00060 ;  Compute A1*D(n-1)
                00061 ;
                00062  MOVFP16  Dn_1,AARG      ; AARG = D(n-2) = multiplier
                00063  MOVFP16  Ax1,BARG       ; BARG = A2 = multiplicand
                00064         call    DblMult          ; (ACCd,ACCc) = A1*D(n-1)
                00065 ;
                00066 ; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
                00067 ;       multiplications already done, so simply perform a 32 bit add
                00068 ;       of previously obtained multiplication results
                00069 ;
                00070   ADD32   DPX,ACC ; ACC = A1*D(n-1) + A2*D(n-2) + (output of 1st section)
                00071 ;
                00072 ;
                00073 ; save the upper 16 bits  of D(n) from the 32 bit accumulator
                00074 ; left shift the result by 1, to adjust the decimal point after
                00075 ; a Q15*Q15 multiplication
                00076 ;
                00077         rlcf     ACC+B1,w
                00078         rlcf     ACC+B2,w
                00079         movwf     Dn
                00080         rlcf     ACC+B3,w        ; decimal adjust ( mult by 2)
                00081         movwf     Dn+B1
                00082 ;
                00083 ; Compute B2 * [D(n) + D(n-2)]
                00084 ;
                00085         #if B0_EQUALS_B2
                00086
                00087         ADD16ACC   Dn_2,Dn,AARG    ; AARG = Dn + D(n-2) = multiplier
                00088  MOVFP16    Bx0,BARG        ; BARG = A2 = multiplicand
                00089         call       DblMult         ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]
                00090  MOVPF32    DPX,ACC
                00091
                00092         #else
                00093
                00094  MOVFP16    Bx0,BARG
```


© 1997 Microchip Technology Inc.

```
00095  MOVFP16     Dn,AARG
00096          call        DblMult          ; B0*D(n)
00097  MOVPF32     DPX,ACC
00098
00099  MOVFP16     Bx2,BARG
00100  MOVFP16     Dn_2,AARG
00101          call        DblMult          ; B2*D(n-2)
00102          ADD32       DPX,ACC
00103
00104          #endif
00105 ;
00106 ; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
00107 ; This way in the next iteration D(n-2) is equal to the present D(n-1)
00108 ;
00109          movfp       Dn_1,AARG+B0
00110          movpf       AARG+B0,Dn_2     ; Shift down D(n-1)
00111          movfp       Dn_1+B1,AARG+B1
00112          movpf       AARG+B1,Dn_2+B1  ; AARG = D(n-1) = multiplier
00113
00114  MOVFP16    Bx1,BARG          ; BARG = B1 = multiplicand
00115
00116          call        DblMult          ; (ACCd,ACCc) = B1*D(n-1)
00117 ;
00118 ; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
00119 ;                 = B1*D(n-1) + B0*[D(n) + D(n-2)]
00120 ; Since all multiplications are already done, simply perform a
00121 ; 32 bit addition
00122 ;
00123          ADD32       DPX,ACC ; ACC = B1*D(n-1) + B2*D(n-2) + B0*D(n)
00124 ;
00125 ; Shift down D(n) to D(n-1) so that in the next iteration, the new
00126 ; D(n-1) is the present D(n)
00127 ;
00128  MOV16      Dn,Dn_1 ; Shift down D(n) to D(n-1)
00129 ;
00130          ENDM
00131
00132          LIST
00023 ;
00024 ;*****************************************************************
00025 ;                Second Order Direct Form IIR Filter
00026 ;
00027 ;
00028 ; In the code given below, a 4th order IIR Elliptic Lowpass Filter
00029 ; is implemented. Other order filters may be implemented by
00030 ; taking the following example code as a basis.
00031 ;
00032 ;
00033 ;        Program:           IIR.ASM
00034 ;        Revision Date:
00035 ;                           1-13-97      Compatibility with MPASMWIN 1.40
00036 ;
00037 ;*****************************************************************
00038 ;
00039 ;        The specifications of the filter are :
00040 ;
00041 ;        Sampling Frequency = 2.0 Khz
00042 ;
00043 ;        Filter Type = 4th Order Elliptic Lowpass Filter
00044 ;
00045 ;                                  Band1           Band2
00046 ;        Lower Band Edge  0.0              600 Hz
00047 ;        Upper Band Edge           500 Hz          1 Khz
00048 ;        Nominal Gain              1.0       0.0
00049 ;        Nominal Ripple    0.01      0.05
00050 ;Maximum Ripple    0.00906   0.04601
```


© 1997 Microchip Technology Inc.

```
00051 ;       Ripple in dB     0.07830         -26.75
00052 ;
00053 ;  The Filter Co-efficients for the above specifications
00054 ; of the filter are computed as follows :
00055 ;
00056 ;      1st Section :
00057 ;                          A11 = -0.133331
00058 ;                          A12 = 0.167145
00059 ;                          B10 = 0.285431
00060 ;                          B11 = 0.462921
00061 ;                          B12 = 0.285431
00062 ;      2nd Section
00063 ;                          A21 = 0.147827
00064 ;                          A22 = 0.765900
00065 ;                          B20 = 0.698273
00066 ;                          B21 = 0.499908
00067 ;                          B22 = 0.698273
00068 ;
00069 ;
00070 ;       Performance (WORST Case):
00071 ;
00072 ;                          Cycles    = #of Filter Stages*775 + 16
00073 ;                                    = 2*775+16 = 1566 Cycles
00074 ;                                          ( 391 uSec)
00075 ;                                  per each sample. Initialization
00076 ;                                  time after reset is not counted
00077 ;                                  Timing measured with B0_EQUALS_B2
00078 ;                                  set to TRUE (see BIQUAD Macro for
00079 ;                                  explanation)
00080 ;
00081 ;                Program Memory :
00082 ;                                  = 16+ # of FilterStages * (BIQUAD Memory
00083 ;                                            + filter co-efficients)
00084 ;                                            + multiplier
00085 ;                                  = 16+2*(63+5)+274 = 421 locations
00086 ;                                    (excluding initialization)
00087 ;
00088 ;                          RAM usage = 48 file registers
00089 ;                          RAM usage/each additional stage = 16 file regs
00090 ;
00091 ;
00092 ;                          This time is less than 2 Khz (500 uSec),
00093 ;                          which means real time filtering is possible
00094 ;
00095 ;*****************************************************************
00096
00097 ;*****************************************************************
00098 ;
00099 ;
00100         CBLOCK  0
00000000 00101                 BB0,BB1,BB2,BB3
00102         ENDC
00103 ;*****************************************************************
00104 ;
00105     CBLOCK  0x18
00000018 00106         DPX,DPX1,DPX2,DPX3              ; arithmetic accumulator
0000001C 00107         AARG,AARG1,BARG,BARG1           ; multiply arguments
00108     ENDC
00109 ;
00110     CBLOCK
00000020 00111             Dn1,    Dn1_Hi
00000022 00112             Dn1_1,  Dn1_1_Hi
00000024 00113             Dn1_2,  Dn1_2_Hi
00114
00000026 00115             Dn2,    Dn2_Hi
00000028 00116             Dn2_1,  Dn2_1_Hi
```


© 1997 Microchip Technology Inc.

```
0000002A   00117             Dn2_2, Dn2_2_Hi
           00118     ENDC
           00119
           00120     CBLOCK
0000002C   00121             A11, A11_Hi
0000002E   00122             A12, A12_Hi
00000030   00123             B10, B10_Hi
00000032   00124             B11, B11_Hi          ; 1st Section Filter Co-efficients
00000034   00125             B12, B12_Hi
           00126
00000036   00127             A21, A21_Hi
00000038   00128             A22, A22_Hi
0000003A   00129             B20, B20_Hi
0000003C   00130             B21, B21_Hi          ; 2nd Section Filter Co-efficients
0000003E   00131             B22, B22_Hi
           00132     ENDC
           00133
           00134     CBLOCK
00000040   00135             X, X1                ; 16 bits of input stream
00000042   00136             Y, Y1                ; 16 bits of filter output
           00137
00000044   00138             ACC, ACC1, ACC2, ACC3 ; 32 bit accumulator for computations
           00139     ENDC
           00140 ;
00000002   00141 FltStage  .set  2
0000000A   00142 NumCoeff  equ   (5*FltStage)      ; 5 Co-eff per stage
           00143 ;
00000001   00144 F          equ   1
           00145 ;
00000001   00146 LPASS     .set  TRUE
00000000   00147 HPASS     .set  FALSE
00000000   00148 BPASS     .set  FALSE            ; select the desired filter type
00000000   00149 BSTOP     .set  FALSE
           00150 ;
00000001   00151 SIGNED   equ     TRUE             ; Set This To 'TRUE' for signed
           00152 ;                                  ; multiplication and 'FALSE' for unsigned.
           00153 ;
           00154 ;*****************************************************************
           00155 ;               Test Program For Low Pass Filter
           00156 ;*****************************************************************
           00157
0000       00158         ORG     0x0000
           00159 ;
0000       00160 start
0000 E00D  00161         call    initFilter
0001 B000  00162         movlw   0x00
0002 0140  00163         movwf   X
0003 B07F  00164         movlw   0x7f              ; set initial Xn = X(0) = 0x7f00
0004 0141  00165         movwf   X+BB1             ; test for impulse response
           00166 ;
0005       00167 NextPoint
0005 E022  00168         call    IIR_Filter
0006 A442  00169         tlwt    _LOW,Y
0007       00170 tracePoint
0007 AE43  00171         tablwt  _HIGH,0,Y+BB1
0008 0000  00172         nop
0009 2940  00173         clrf    X, F              ; set X(n) = 0 , n <> 0
000A 2941  00174         clrf    X+BB1, F          ; for simulating an Impulse
000B C005  00175         goto    NextPoint
           00176 ;
000C C00C  00177 self    goto    self
           00178 ;
           00179 ;*****************************************************************
           00180 ;
000D       00181 initFilter
           00182 ;
```

```
                00183 ;   At first read the Filter Co-efficients from Prog. Mem to Data RAM
                00184 ;
                00185         #if     LPASS
000D B0C1       00186                 movlw   LOW     _coeff_lpass
000E 010D       00187                 movwf   TBLPTRL
000F B001       00188                 movlw   HIGH    _coeff_lpass
0010 010E       00189                 movwf   TBLPTRH
                00190         #endif
                00191         #if     HPASS
                00192                 movlw   LOW     _coeff_hpass
                00193                 movwf   TBLPTRL
                00194                 movlw   HIGH    _coeff_hpass
                00195                 movwf   TBLPTRH
                00196         #endif
                00197         #if     BPASS
                00198                 movlw   LOW     _coeff_bpass
                00199                 movwf   TBLPTRL
                00200                 movlw   HIGH    _coeff_bpass
                00201                 movwf   TBLPTRH
                00202         #endif
                00203         #if     BSTOP
                00204                 movlw   LOW     _coeff_bstop
                00205                 movwf   TBLPTRL
                00206                 movlw   HIGH    _coeff_bstop
                00207                 movwf   TBLPTRH
                00208         #endif
                00209
                00210 ;
0011 B02C       00211         movlw   A11
0012 0101       00212         movwf   FSR0
0013 8404       00213         bsf     ALUSTA,FS0
0014 8D04       00214         bcf     ALUSTA,FS1              ; auto increment
                00215 ;
                00216 ; Read Filter Co-efficients from Program Memory
                00217 ;
0015 B00A       00218         movlw   NumCoeff
0016 A92C       00219         tablrd  _LOW,_INC,A11         ; garbage
0017            00220 NextCoeff
0017 A000       00221         tlrd    _LOW,INDF0
0018 AB00       00222         tablrd  _HIGH,_INC,INDF0
0019 170A       00223         decfsz  WREG, F
001A C017       00224         goto    NextCoeff
                00225 ;
                00226 ; Initialize "Dn"s to zero
                00227 ;
001B B020       00228         movlw   Dn1
001C 0101       00229         movwf   FSR0
001D B00C       00230         movlw   6*FltStage
001E            00231 NextClr
001E 2900       00232         clrf    INDF0, F
001F 170A       00233         decfsz  WREG, F
0020 C01E       00234         goto    NextClr
                00235 ;
0021 0002       00236         return
                00237 ;
                00238 ;*****************************************************************
                00239 ;                       1st Cascade Section
                00240 ;*****************************************************************
                00241 ;
0022            00242 IIR_Filter
                00243 ;
                00244 ; Compute  D(n) = X(n) + A1*D(n-1) + A2*D(n-2)
                00245 ;          Since the filter constants are computated in Q15 format,
                00246 ;          X(n) must be multiplied by 2**15 and then added to the
                00247 ;          other terms.
                00248 ;
```


© 1997 Microchip Technology Inc.

```
                00249 ; Move Input to accumulator after proper scaling
                00250 ;
0022 8804       00251         bcf     ALUSTA,C
0023 1941       00252         rrcf    X+BB1, F
0024 1940       00253         rrcf    X, F
0025 290A       00254         clrf    WREG, F     ; Scale the input X
0026 190A       00255         rrcf    WREG, F
0027 0145       00256         movwf   ACC+BB1
0028 6A40       00257         movfp   X,WREG
0029 0146       00258         movwf   ACC+BB2
002A 6A41       00259         movfp   X+BB1,WREG
002B 0147       00260         movwf   ACC+BB3     ; ACC = scaled input : X*(2**15)
                00261 ;
                00262 ; 1st Biquad filter section
                00263 ;
                00264         BIQUAD  A11,A12,B10,B11,Dn1,Dn1_1,Dn1_2,1
                    M
                    M ;
                    M ; Compute Ax2*D(n-2)
                    M ;
                    M   MOVFP16 Dn1_2,AARG        ; D(n-2) = multiplier
                    M
002C 7C24           M   MOVFP   Dn1_2+B0,AARG+B0 ; move A(B0) to B(B0)
002D 7D25           M   MOVFP   Dn1_2+B1,AARG+B1 ; move A(B1) to B(B1)
                    M
                    M   MOVFP16 A12,BARG          ; A2 = multiplicand
                    M
002E 7E2E           M   MOVFP   A12+B0,BARG+B0    ; move A(B0) to B(B0)
002F 7F2F           M   MOVFP   A12+B1,BARG+B1    ; move A(B1) to B(B1)
                    M
0030 E0AF           M   call    DblMult           ; (ACCd,ACCc) = A2*D(n-2)
                    M ;
                    M ; Add product to output of 1st section
                    M ; Save result in 32 bit Accumulator
                    M ;
                    M   ADD32   DPX,ACC
                    M
0031 6A18           M   MOVFP   DPX+B0,WREG       ; get lowest byte of a into w
0032 0F44           M   ADDWF   ACC+B0, F         ; add lowest byte of b, save in b(B0)
0033 6A19           M   MOVFP   DPX+B1,WREG       ; get 2nd byte of a into w
0034 1145           M   ADDWFC  ACC+B1, F         ; add 2nd byte of b, save in b(B1)
0035 6A1A           M   MOVFP   DPX+B2,WREG       ; get 3rd byte of a into w
0036 1146           M   ADDWFC  ACC+B2, F         ; add 3rd byte of b, save in b(B2)
0037 6A1B           M   MOVFP   DPX+B3,WREG       ; get 4th byte of a into w
0038 1147           M   ADDWFC  ACC+B3, F         ; add 4th byte of b, save in b(B3)
                    M
                    M ;
                    M ;  Compute A1*D(n-1)
                    M ;
                    M   MOVFP16  Dn1_1,AARG       ; AARG = D(n-2) = multiplier
                    M
0039 7C22           M   MOVFP   Dn1_1+B0,AARG+B0  ; move A(B0) to B(B0)
003A 7D23           M   MOVFP   Dn1_1+B1,AARG+B1  ; move A(B1) to B(B1)
                    M
                    M   MOVFP16  A11,BARG         ; BARG = A2 = multiplicand
                    M
003B 7E2C           M   MOVFP   A11+B0,BARG+B0    ; move A(B0) to B(B0)
003C 7F2D           M   MOVFP   A11+B1,BARG+B1    ; move A(B1) to B(B1)
                    M
003D E0AF           M   call    DblMult           ; (ACCd,ACCc) = A1*D(n-1)
                    M ;
                    M ; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
                    M ;       multiplications already done, so simply perform a 32 bit add
                    M ;       of previously obtained multiplication results
                    M ;
                    M   ADD32   DPX,ACC                ; ACC = A1*D(n-1) + A2*D(n-2) +
```

```
                M                                       ;           (output of 1st section)
003E 6A18       M   MOVFP   DPX+B0,WREG                 ; get lowest byte of a into w
003F 0F44       M   ADDWF   ACC+B0, F                   ; add lowest byte of b, save in b(B0)
0040 6A19       M   MOVFP   DPX+B1,WREG                 ; get 2nd byte of a into w
0041 1145       M   ADDWFC  ACC+B1, F                   ; add 2nd byte of b, save in b(B1)
0042 6A1A       M   MOVFP   DPX+B2,WREG                 ; get 3rd byte of a into w
0043 1146       M   ADDWFC  ACC+B2, F                   ; add 3rd byte of b, save in b(B2)
0044 6A1B       M   MOVFP   DPX+B3,WREG                 ; get 4th byte of a into w
0045 1147       M   ADDWFC  ACC+B3, F                   ; add 4th byte of b, save in b(B3)
                M
                M ;
                M ;
                M ; save the upper 16 bits  of D(n) from the 32 bit accumulator
                M ; left shift the result by 1, to adjust the decimal point after
                M ; a Q15*Q15 multiplication
                M ;
0046 1A45       M   rlcf    ACC+B1,w
0047 1A46       M   rlcf    ACC+B2,w
0048 0120       M   movwf   Dn1
0049 1A47       M   rlcf    ACC+B3,w                ; decimal adjust ( mult by 2)
004A 0121       M   movwf   Dn1+B1
                M ;
                M ; Compute B2 * [D(n) + D(n-2)]
                M ;
                M   #if B0_EQUALS_B2
                M
                M    ADD16ACC  Dn1_2,Dn1,AARG     ; AARG = Dn + D(n-2) = multiplier
                M
004B 6A24       M    movfp   Dn1_2+B0,WREG
004C 0E20       M    addwf   Dn1+B0,w
004D 011C       M    movwf   AARG+B0
004E 6A25       M    movfp   Dn1_2+B1,WREG
004F 1021       M    addwfc  Dn1+B1,w
0050 011D       M    movwf   AARG+B1
                M
                M    MOVFP16  B10,BARG            ; BARG = A2 = multiplicand
                M
0051 7E30       M    MOVFP   B10+B0,BARG+B0       ; move A(B0) to B(B0)
0052 7F31       M    MOVFP   B10+B1,BARG+B1       ; move A(B1) to B(B1)
                M
0053 E0AF       M    call    DblMult              ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]
                M    MOVPF32  DPX,ACC
                M
0054 5844       M    MOVPF   DPX+B0,ACC+B0        ; move A(B0) to B(B0)
0055 5945       M    MOVPF   DPX+B1,ACC+B1        ; move A(B1) to B(B1)
0056 5A46       M    MOVPF   DPX+B2,ACC+B2        ; move A(B2) to B(B2)
0057 5B47       M    MOVPF   DPX+B3,ACC+B3        ; move A(B3) to B(B3)
                M
                M
                M    #else
                M
                M     MOVFP16 B10,BARG
                M     MOVFP16 Dn1,AARG
                M     call    DblMult              ; B0*D(n)
                M     MOVPF32 DPX,ACC
                M
                M     MOVFP16 Bx2,BARG
                M     MOVFP16 Dn1_2,AARG
                M     call    DblMult              ; B2*D(n-2)
                M     ADD32   DPX,ACC
                M
                M     #endif
                M ;
                M ; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
                M ; This way in the next iteration D(n-2) is equal to the present D(n-1)
                M ;
```


© 1997 Microchip Technology Inc.

```
0058 7C22        M    movfp    Dn1_1,AARG+B0
0059 5C24        M    movpf    AARG+B0,Dn1_2        ; Shift down D(n-1)
005A 7D23        M    movfp    Dn1_1+B1,AARG+B1
005B 5D25        M    movpf    AARG+B1,Dn1_2+B1     ; AARG = D(n-1) = multiplier
                 M
                 M    MOVFP16  B11,BARG             ; BARG = B1 = multiplicand
                 M
005C 7E32        M    MOVFP    B11+B0,BARG+B0       ; move A(B0) to B(B0)
005D 7F33        M    MOVFP    B11+B1,BARG+B1       ; move A(B1) to B(B1)
                 M
                 M
005E E0AF        M    call     DblMult              ; (ACCd,ACCc) = B1*D(n-1)
                 M ;
                 M ; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
                 M ;                  = B1*D(n-1) + B0*[D(n) + D(n-2)]
                 M ; Since all multiplications are already done, simply perform a
                 M ; 32 bit addition
                 M ;
                 M    ADD32    DPX,ACC ; ACC = B1*D(n-1) + B2*D(n-2) + B0*D(n)
                 M
005F 6A18        M    MOVFP    DPX+B0,WREG          ; get lowest byte of a into w
0060 0F44        M    ADDWF    ACC+B0, F            ; add lowest byte of b, save in b(B0)
0061 6A19        M    MOVFP    DPX+B1,WR            ; get 2nd byte of a into w
0062 1145        M    ADDWFC   ACC+B1, F            ; add 2nd byte of b, save in b(B1)
0063 6A1A        M    MOVFP    DPX+B2,WREG          ; get 3rd byte of a into w
0064 1146        M    ADDWFC   ACC+B2, F            ; add 3rd byte of b, save in b(B2)
0065 6A1B        M    MOVFP    DPX+B3,WREG          ; get 4th byte of a into w
0066 1147        M    ADDWFC   ACC+B3, F            ; add 4th byte of b, save in b(B3)
                 M
                 M ;
                 M ; Shift down D(n) to D(n-1) so that in the next iteration, the new
                 M ; D(n-1) is the present D(n)
                 M ;
                 M    MOV16    Dn1,Dn1_1 ; Shift down D(n) to D(n-1)
                 M
0067 6A20        M    MOVFP    Dn1+B0,WREG          ; get byte of a into w
0068 0122        M    MOVWF    Dn1_1+B0             ; move to b(B0)
0069 6A21        M    MOVFP    Dn1+B1,WREG          ; get byte of a into w
006A 0123        M    MOVWF    Dn1_1+B1             ; move to b(B1)
                 M
                 M ;
              00265 ;
              00266 ; 2nd Biquad filter section
              00267 ;
              00268    BIQUAD   A21,A22,B20,B21,Dn2,Dn2_1,Dn2_2,2
                 M
                 M ;
                 M ; Compute Ax2*D(n-2)
                 M ;
                 M    MOVFP16  Dn2_2,AARG           ; D(n-2) = multiplier
                 M
006B 7C2A        M    MOVFP    Dn2_2+B0,AARG+B0     ; move A(B0) to B(B0)
006C 7D2B        M    MOVFP    Dn2_2+B1,AARG+B1     ; move A(B1) to B(B1)
                 M
                 M    MOVFP16  A22,BARG             ; A2 = multiplicand
                 M
006D 7E38        M    MOVFP    A22+B0,BARG+B0       ; move A(B0) to B(B0)
006E 7F39        M    MOVFP    A22+B1,BARG+B1       ; move A(B1) to B(B1)
                 M
006F E0AF        M    call     DblMult              ; (ACCd,ACCc) = A2*D(n-2)
                 M ;
                 M ; Add product to output of 1st section
                 M ; Save result in 32 bit Accumulator
                 M ;
                 M    ADD32    DPX,ACC
                 M
```


© 1997 Microchip Technology Inc.

```
0070 6A18        M    MOVFP   DPX+B0,WREG          ; get lowest byte of a into w
0071 0F44        M    ADDWF   ACC+B0, F            ; add lowest byte of b, save in b(B0)
0072 6A19        M    MOVFP   DPX+B1,WREG          ; get 2nd byte of a into w
0073 1145        M    ADDWFC  ACC+B1, F            ; add 2nd byte of b, save in b(B1)
0074 6A1A        M    MOVFP   DPX+B2,WREG          ; get 3rd byte of a into w
0075 1146        M    ADDWFC  ACC+B2, F            ; add 3rd byte of b, save in b(B2)
0076 6A1B        M    MOVFP   DPX+B3,WREG          ; get 4th byte of a into w
0077 1147        M    ADDWFC  ACC+B3, F            ; add 4th byte of b, save in b(B3)
                 M
                 M ;
                 M ;  Compute A1*D(n-1)
                 M ;
                 M    MOVFP16  Dn2_1,AARG          ; AARG = D(n-2) = multiplier
                 M
0078 7C28        M    MOVFP   Dn2_1+B0,AARG+B0     ; move A(B0) to B(B0)
0079 7D29        M    MOVFP   Dn2_1+B1,AARG+B1     ; move A(B1) to B(B1)
                 M
                 M    MOVFP16  A21,BARG            ; BARG = A2 = multiplicand
                 M
007A 7E36        M    MOVFP   A21+B0,BARG+B0       ; move A(B0) to B(B0)
007B 7F37        M    MOVFP   A21+B1,BARG+B1       ; move A(B1) to B(B1)
                 M
007C E0AF        M    call    DblMult              ; (ACCd,ACCc) = A1*D(n-1)
                 M ;
                 M ; Compute A1*D(n-1) + A2*D(n-2) + output of previous section
                 M ;       multiplications already done, so simply perform a 32 bit add
                 M ;       of previously obtained multiplication results
                 M ;
                 M   ADD32   DPX,ACC              ; ACC = A1*D(n-1) + A2*D(n-2) +
                 M                                ;              (output of 1st section)
007D 6A18        M   MOVFP   DPX+B0,WREG          ; get lowest byte of a into w
007E 0F44        M   ADDWF   ACC+B0, F            ; add lowest byte of b, save in b(B0)
007F 6A19        M   MOVFP   DPX+B1,WREG          ; get 2nd byte of a into w
0080 1145        M   ADDWFC  ACC+B1, F            ; add 2nd byte of b, save in b(B1)
0081 6A1A        M   MOVFP   DPX+B2,WREG          ; get 3rd byte of a into w
0082 1146        M   ADDWFC  ACC+B2, F            ; add 3rd byte of b, save in b(B2)
0083 6A1B        M   MOVFP   DPX+B3,WREG          ; get 4th byte of a into w
0084 1147        M   ADDWFC  ACC+B3, F            ; add 4th byte of b, save in b(B3)
                 M
                 M ;
                 M ;
                 M ; save the upper 16 bits  of D(n) from the 32 bit accumulator
                 M ; left shift the result by 1, to adjust the decimal point after
                 M ; a Q15*Q15 multiplication
                 M ;
0085 1A45        M   rlcf    ACC+B1,w
0086 1A46        M   rlcf    ACC+B2,w
0087 0126        M   movwf   Dn2
0088 1A47        M   rlcf    ACC+B3,w             ; decimal adjust ( mult by 2)
0089 0127        M   movwf   Dn2+B1
                 M ;
                 M ; Compute B2 * [D(n) + D(n-2)]
                 M ;
                 M   #if B0_EQUALS_B2
                 M
                 M   ADD16ACC Dn2_2,Dn2,AARG      ; AARG = Dn + D(n-2) = multiplier
                 M
008A 6A2A        M   movfp   Dn2_2+B0,WREG
008B 0E26        M   addwf   Dn2+B0,w
008C 011C        M   movwf   AARG+B0
008D 6A2B        M   movfp   Dn2_2+B1,WREG
008E 1027        M   addwfc  Dn2+B1,w
008F 011D        M   movwf   AARG+B1
                 M
                 M   MOVFP16 B20,BARG            ; BARG = A2 = multiplicand
                 M
```


© 1997 Microchip Technology Inc.

```
0090 7E3A       M   MOVFP   B20+B0,BARG+B0      ; move A(B0) to B(B0)
0091 7F3B       M   MOVFP   B20+B1,BARG+B1      ; move A(B1) to B(B1)
                M
0092 E0AF       M   call    DblMult             ; (ACCd,ACCc) = B2*[D(n)+D(n-2)]
                M   MOVPF32 DPX,ACC
                M
0093 5844       M   MOVPF   DPX+B0,ACC+B0       ; move A(B0) to B(B0)
0094 5945       M   MOVPF   DPX+B1,ACC+B1       ; move A(B1) to B(B1)
0095 5A46       M   MOVPF   DPX+B2,ACC+B2       ; move A(B2) to B(B2)
0096 5B47       M   MOVPF   DPX+B3,ACC+B3       ; move A(B3) to B(B3)
                M
                M
                M   #else
                M
                M   MOVFP16 B20,BARG
                M   MOVFP16 Dn2,AARG
                M   call    DblMult             ; B0*D(n)
                M   MOVPF32 DPX,ACC
                M
                M   MOVFP16 Bx2,BARG
                M   MOVFP16 Dn2_2,AARG
                M   call    DblMult             ; B2*D(n-2)
                M   ADD32   DPX,ACC
                M
                M   #endif
                M ;
                M ; Shift down D(n-1) to D(n-2) after D(n-2) usage is no longer required.
                M ; This way in the next iteration D(n-2) is equal to the present D(n-1)
                M ;
0097 7C28       M  movfp    Dn2_1,AARG+B0
0098 5C2A       M  movpf    AARG+B0,Dn2_2       ; Shift down D(n-1)
0099 7D29       M  movfp    Dn2_1+B1,AARG+B1
009A 5D2B       M  movpf    AARG+B1,Dn2_2+B1    ; AARG = D(n-1) = multiplier
                M
                M  MOVFP16  B21,BARG            ; BARG = B1 = multiplicand
                M
009B 7E3C       M  MOVFP   B21+B0,BARG+B0       ; move A(B0) to B(B0)
009C 7F3D       M  MOVFP   B21+B1,BARG+B1       ; move A(B1) to B(B1)
                M
                M
009D E0AF       M  call    DblMult              ; (ACCd,ACCc) = B1*D(n-1)
                M ;
                M ; Compute Output Y = B1*D(n-1) + B2*D(n-2) + B0*D(n)
                M ;                  = B1*D(n-1) + B0*[D(n) + D(n-2)]
                M ; Since all multiplications are already done, simply perform a
                M ; 32 bit addition
                M ;
                M   ADD32   DPX,ACC             ; ACC = B1*D(n-1) + B2*D(n-2) + B0*D(n)
                M
009E 6A18       M   MOVFP   DPX+B0,WREG         ; get lowest byte of a into w
009F 0F44       M   ADDWF   ACC+B0, F           ; add lowest byte of b, save in b(B0)
00A0 6A19       M   MOVFP   DPX+B1,WREG         ; get 2nd byte of a into w
00A1 1145       M   ADDWFC  ACC+B1, F           ; add 2nd byte of b, save in b(B1)
00A2 6A1A       M   MOVFP   DPX+B2,WREG         ; get 3rd byte of a into w
00A3 1146       M   ADDWFC  ACC+B2, F           ; add 3rd byte of b, save in b(B2)
00A4 6A1B       M   MOVFP   DPX+B3,WREG         ; get 4th byte of a into w
00A5 1147       M   ADDWFC  ACC+B3, F           ; add 4th byte of b, save in b(B3)
                M
                M ;
                M ; Shift down D(n) to D(n-1) so that in the next iteration, the new
                M ; D(n-1) is the present D(n)
                M ;
                M   MOV16   Dn2,Dn2_1           ; Shift down D(n) to D(n-1)
                M
00A6 6A26       M   MOVFP   Dn2+B0,WREG         ; get byte of a into w
00A7 0128       M   MOVWF   Dn2_1+B0            ; move to b(B0)
```

4

```
00A8 6A27        M   MOVFP   Dn2+B1,WREG          ; get byte of a into w
00A9 0129        M   MOVWF   Dn2_1+B1             ; move to b(B1)
                 M
                 M ;
                00269 ;
                00270 ; The filter output is now computed
                00271 ; Save the Upper 16 Bits of 32 bit Accumulator into Y after
                00272 ; adjusting the decimal point
                00273 ;
                00274
                00275   MOV16   ACC+BB2,Y
                    M
00AA 6A46           M   MOVFP   ACC+BB2+B0,WREG      ; get byte of a into w
00AB 0142           M   MOVWF   Y+B0                 ; move to b(B0)
00AC 6A47           M   MOVFP   ACC+BB2+B1,WREG      ; get byte of a into w
00AD 0143           M   MOVWF   Y+B1                 ; move to b(B1)
                    M
                00276 ;
00AE 0002       00277   return                       ; Output Y(n) computed
                00278 ;
                00279 ;*****************************************************************
                00280 ;  Set SIGNED/UNSIGNED Flag Before Including 17c42MPY.mac
                00281 ;
                00282   #include        <17c42MPY.mac>
                00178   LIST
                00283
                00284 ;*****************************************************************
                00285 ;  Low Pass Filter Co-efficients
                00286 ;
                00287 ;
                00288 ;                ELLIPTIC     LOWPASS FILTER
                00289 ;
                00290 ;                FILTER ORDER =    4
                00291 ;                Sampling frequency =   2.000  KiloHertz
                00292 ;
                00293 ;                        BAND  1        BAND  2
                00294 ;
                00295 ;
                00296 ; LOWER BAND EDGE         .00000         .60000
                00297 ; UPPER BAND EDGE         .50000        1.00000
                00298 ; NOMINAL GAIN           1.00000         .00000
                00299 ; NOMINAL RIPPLE          .01000         .05000
                00300 ; MAXIMUM RIPPLE          .00906         .04601
                00301 ; RIPPLE IN dB            .07830      -26.74235
                00302 ;
                00303 ;  I    A(I,1)        A(I,2)        B(I,0)        B(I,1)        B(I,2)
                00304 ;
                00305 ;  1    -.133331       .167145       .285431       .462921       .285431
                00306 ;       .147827       .765900       .698273       .499908       .698273
                00307 ;
                00308
01C1            00309 _coeff_lpass            ; co-efficients for 1st Cascade section
01C1 1111       00310         data     4369           ; -A11
01C2 EA9B       00311         data     -5477          ; -A12
01C3 2489       00312         data     9353           ; B10
01C4 3B41       00313         data     15169          ; B11
01C5 2489       00314         data     9353           ; B12
                00315                         ; co-efficients for 2nd Cascade section
01C6 ED14       00316         data    -4844           ; -A21
01C7 9DF7       00317         data    -25097          ; -A22
01C8 5961       00318         data     22881          ; B20
01C9 3FFD       00319         data     16381          ; B21
01CA 5961       00320         data     22881          ; B22
                00321 ;
                00322 ;*****************************************************************
                00323 ;
```


© 1997 Microchip Technology Inc.

```
                00324 ;
                00325 ;*********************************************************************
                00326 ;  High Pass Filter Co-efficients
                00327 ;
                00328 ;
                00329 ;                 ELLIPTIC    HIGHPASS FILTER
                00330 ;
                00331 ;                 FILTER ORDER =    4
                00332 ;                 Sampling frequency =   2.000  KiloHertz
                00333 ;
                00334 ;                         BAND  1        BAND  2
                00335 ;
                00336 ;
                00337 ; LOWER BAND EDGE          .00000         .50000
                00338 ; UPPER BAND EDGE          .40000        1.00000
                00339 ; NOMINAL GAIN             .00000        1.00000
                00340 ; NOMINAL RIPPLE           .04000         .02000
                00341 ; MAXIMUM RIPPLE           .03368         .01686
                00342 ; RIPPLE IN dB          -29.45335         .14526
                00343 ;
                00344 ;  I      A(I,1)        A(I,2)        B(I,0)        B(I,1)        B(I,2)
                00345 ;
                00346 ;  1     .276886       .195648       .253677      -.411407       .253677
                00347 ;  2    -.094299       .780396       .678650      -.485840       .678650
                00348 ;
                00349 ;
01CB            00350 _coeff_hpass            ; co-efficients for 1st Cascade section
01CB DC8F       00351         data     -9073          ; -A11
01CC E6F5       00352         data     -6411          ; -A12
01CD 2079       00353         data     8313           ; B10
01CE CB57       00354         data     -13481         ; B11
01CF 2079       00355         data     8313           ; B12
                00356 ;                       ; co-efficients for 2nd Cascade section
01D0 0C12       00357         data    3090            ; -A21
01D1 9C1C       00358         data    -25572          ; -A22
01D2 56DE       00359         data    22238           ; B20
01D3 C1D0       00360         data    -15920          ; B21
01D4 56DE       00361         data    22238           ; B22
                00362 ;
                00363 ;*********************************************************************
                00364 ;
                00365 ;*********************************************************************
                00366 ;  Band Pass Filter Co-efficients
                00367 ;
                00368 ;
                00369 ;                 ELLIPTIC     BANDPASS FILTER
                00370 ;
                00371 ;                 FILTER ORDER =    4
                00372 ;                 Sampling frequency =   2.000  KiloHertz
                00373 ;
                00374 ;                         BAND  1        BAND  2        BAND  3
                00375 ;
                00376 ;
                00377 ; LOWER BAND EDGE          .00000         .30000         .90000
                00378 ; UPPER BAND EDGE          .10000         .70000        1.00000
                00379 ; NOMINAL GAIN             .00000        1.00000         .00000
                00380 ; NOMINAL RIPPLE           .05000         .05000         .05000
                00381 ; MAXIMUM RIPPLE           .03644         .03867         .03641
                00382 ; RIPPLE IN dB          -28.76779         .32956      -28.77647
                00383 ;
                00384 ;
                00385 ;  I      A(I,1)        A(I,2)        B(I,0)        B(I,1)        B(I,2)
                00386 ;
                00387 ;  1    -.936676       .550568       .444000      -.865173       .444000
                00388 ;  2     .936707       .550568       .615540      1.199402       .615540
                00389 ;
```

```
                00390
01D5            00391 _coeff_bpass                  ; co-efficients for 1st Cascade section
01D5 3BF2       00392         data     30693/2          ; -A11
01D6 DCC4       00393         data     -18041/2         ; -A12
01D7 1C6A       00394         data     14549/2          ; B10
01D8 C8A1       00395         data     -28350/2         ; B11
01D9 1C6A       00396         data     14549/2          ; B12
                00397 ;                               ; co-efficients for 2nd Cascade section
01DA C40D       00398         data    -30694/2          ; -A21
01DB DCC4       00399         data    -18041/2          ; -A22
01DC 2765       00400         data     20170/2          ; B20
01DD 4CC3       00401         data     39302/2          ; B21
01DE 2765       00402         data     20170/2          ; B22
                00403 ;
                00404 ;*****************************************************************
                00405 ;
                00406 ;*****************************************************************
                00407 ;  Band Stop Filter Co-efficients
                00408 ;
                00409 ;
                00410 ;                       ELLIPTIC    BANDSTOP FILTER
                00411 ;
                00412 ;               FILTER ORDER =    4
                00413 ;               Sampling frequency =   2.000  KiloHertz
                00414 ;
                00415 ;                        BAND  1        BAND  2        BAND  3
                00416 ;
                00417 ;
                00418 ;LOWER BAND EDGE          .00000         .45000         .70000
                00419 ;UPPER BAND EDGE          .30000         .55000        1.00000
                00420 ;NOMINAL GAIN            1.00000         .00000        1.00000
                00421 ;NOMINAL RIPPLE           .05000         .05000         .05000
                00422 ;MAXIMUM RIPPLE           .03516         .03241         .03517
                00423 ;RIPPLE IN dB             .30015      -29.78523         .30027
                00424 ;
                00425 ;
                00426 ;  I      A(I,1)        A(I,2)        B(I,0)        B(I,1)        B(I,2)
                00427 ;
                00428 ;  1     .749420       .583282       .392685       .087936       .392685
                00429 ;  2     -.749390      .583282      1.210022      -.270935      1.210022
                00430 ;
01DF            00431 _coeff_bstop                  ; co-efficients for 1st Cascade section
01DF D00A       00432         data     -24557/2         ; -A11
01E0 DAAC       00433         data     -19113/2         ; -A12
01E1 1922       00434         data     12868/2          ; B10
01E2 05A0       00435         data     2881/2           ; B11
01E3 1922       00436         data     12868/2          ; B12
                00437                                 ; co-efficients for 2nd Cascade section
01E4 2FF6       00438         data     24556/2          ; -A21
01E5 DAAC       00439         data     -19113/2         ; -A22
01E6 4D71       00440         data     39650/2          ; B20
01E7 EEA9       00441         data     -8878/2          ; B21
01E8 4D71       00442         data     39650/2          ; B22
                00443 ;
                00444 ;*****************************************************************
                00445
                00446         END
```


© 1997 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXX------- ----------------

All other memory blocks unused.

Program Memory Words Used:   489


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN542

Implementation of Fast Fourier Transforms

Author: *Amar Palacherla*
Microchip Technology Inc.

INTRODUCTION

Fourier transforms are one of the fundamental operations in signal processing. In digital computations, Discrete Fourier Transforms (DFT) are used to describe, represent, and analyze discrete-time signals. However, direct implementation of DFT is computationally very inefficient. Of the various available high speed algorithms to compute DFT, the Cooley-Tukey algorithm is the simplest and most commonly used. These efficient algorithms, used to compute DFTs, are called Fast Fourier Transforms (FFTs).

This application note provides the source code to compute FFTs using a PIC17C42. The theory behind the FFT algorithms is well established and described in literature and hence not described in this application note. A Radix-2 Cooley-Tukey FFT is implemented with no limits on the length of the FFT. The length is only limited by the amount of available program memory space. All computations are performed using double precision arithmetic.

IMPLEMENTATION

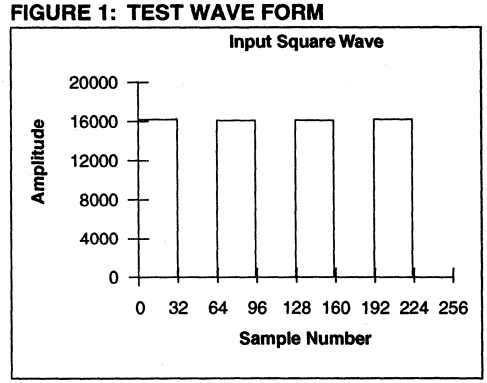
Since the PIC17C42 has only 232 x 8 general purpose RAM (equivalent of 116 x 16), at most a 32-point FFT (16-bit REAL & IMAGINARY data) can be implemented using on-chip RAM. To compute higher point FFTs, the data can be stored in the program memory space of the PIC17C42. The PIC17C42 has instructions (`TABLRD` & `TABLWT`) to transfer data between program memory space and on-chip file registers. In extended microcontroller mode, the PIC17C42 has 2K x 16 (0000h:07FFh) on-chip program memory space and is capable of addressing 62K x 16 (0800h:0FFFFh) of external program memory space. In this mode, the code (in this case, the FFT code) may reside on the on-chip EPROM and the data to be analyzed may be stored in external RAM (up to 62K). A suggested method of connecting external RAM (appropriate EEPROMs may also be used) is shown in Figure 3.

If the PIC17C42 is used in extended microcontroller mode and if all the code resides on-chip, then the cost may further be reduced by using only one external SRAM instead of two. The block diagram is shown in Figure 4. The 16-bit data stored in the external RAM is organized as low byte followed by high byte. To achieve this, the code presented in this application note needs minor modifications, especially where `TABLRD` and `TABLWT` instructions are used. Address indexing must be incremented by two since two reads/writes must be performed to access a 16-bit data.

The FFT is implemented with Decimation In Frequency. Thus the input data, before calling the FFT routine (`R2FFT`), should be in normal order and the transformed data should be in scrambled order. The original data is overwritten by the transformed data to conserve memory. This is achieved by use of in-place calculations. These in-place calculations cause the order of the DFT terms to be permuted. So at the end of the transform, all of the data needs to be unscrambled to get the right order of the DFT terms. In some applications the order of terms is not necessary. Keeping this in mind, the unscrambling code is written as a separate subroutine (`Unscramble`) and may be called if necessary.

Before implementing the FFT using a PIC17C42, a C program was written and tested. This high level programming helps in writing the assembly code and the results of both programs can be compared while debugging the assembly code. The C source code for the Radix-2 FFT is shown in Appendix A. The assembly code source file of the FFT program is shown in Appendix B. For a listing of the header file `17C42.h` and the macro definition file `17C42.mac` please refer to Appendices C and D respectively of application note AN540.

FIGURE 1: TEST WAVE FORM



© 1997 Microchip Technology Inc.

TESTING

The assembly code was developed and debugged using Microchip's PICMASTER In-Circuit Emulator System. A main program generates a test pattern (like a square wave) and calls FFT routines `R2FFT` and `Unscramble`. After the DFT terms are computed, the results are captured into PICMASTER's real-time trace buffer by putting a trace point on a dummy `TABLRD` instruction and capturing only the 2nd cycle (data cycle of `TABLRD`) of the instruction. The data from the trace buffer was hot linked to a Microsoft Excel spread sheet using DDE and then the graphs were plotted and analyzed.

The code was tested on various waveforms (a rectangular pulse, a triangular wave, square wave and a sine wave) using FFT lengths of 64, 256 and 1024. The results of a 256 point FFT on a square wave is shown below. The test waveform is shown in Figure 1 and it's frequency spectrum computed by the PIC17C42 is shown in Figure 2. As expected, the spectra appears at the odd harmonics of the input waveform's fundamental frequency (At N*256/64, N = 0,1,3,5,..).

PERFORMANCE

The performance of FFTs using a PIC17C42 is quite impressive for an 8-bit machine with no hardware multiplier. Also note that all computations are performed using double precision arithmetic (16- and 32-bit) which is the case for most of the low end DSPs. Table 1 provides the real-time performance in total number of Instruction cycles for both the `R2FFT` and `Unscramble` routines using 64, 256 and 1024 point FFTs. Note that the timings are in a worst case situation and in general will be a lot better than shown in the table. The worst case situation arises because the 16 x 16 software multiplier (`DblMult`) does not have uniform timing and depends on the input data. The worst case timing of the multiplier is used in computing it's performance.

FIGURE 2: FFT (MAGNITUDE SPECTRUM) OF FIGURE 1 COMPUTED BY A PIC17C42

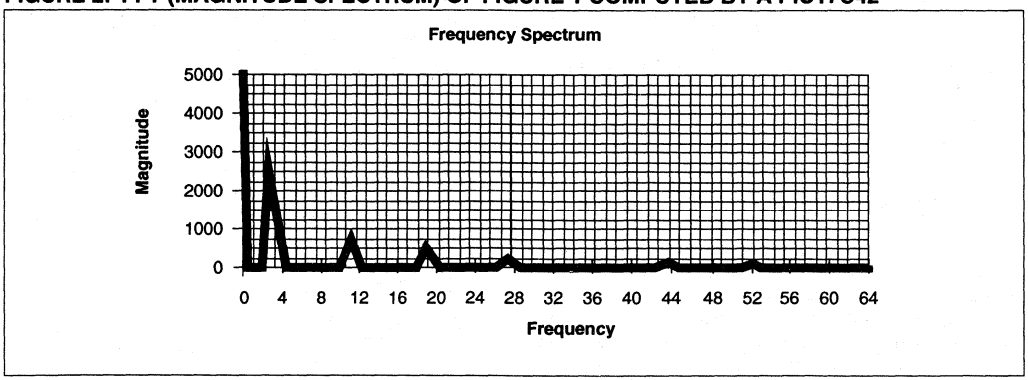


TABLE 1: WORST CASE PERFORMANCE OF FFT IN INSTRUCTION CYCLES AND REAL-TIME @ 25 MHz

N (FFT Length)	64 Point	256 Point	1024 Point
R2FFT	34116 + 768* Mult = 171588	178024 + 4096* Mult = 911208	878752 + 20480* Mult = 4544672
Unscramble	5143	22495	93525
Total	176731 (28.28 ms)	933703 (149.39 ms)	4638197 (742.11 ms)

© 1997 Microchip Technology Inc.

Table 2 shows the Program Memory and Data RAM requirements for an N Point FFT. The multiplier routine and other general purpose macro requirements are included in the memory requirements. The speed performance for the square wave test differs from Table 1, since "worst case timings" is not used and reflects a more reasonable data.

FFT APPLICATIONS

Although the FFT does not find a place in many microcontroller applications, it is very useful in providing a benchmark of the processor. As can be seen from Table 2, the performance is very satisfactory, considering the fact that the PIC17C42 is a Microcontroller and not a DSP. Also it should be borne in mind that all computations are performed in 16/32 bit arithmetic and that the PIC17C42 is a low-cost 8-bit device unlike DSPs which are relatively expensive.

In applications such as Instrumentation, where real-time FFT computation is not required, a PIC17C42 can be used as a single chip solution instead of a microcontroller and a Digital Signal Processor.

Suggested Reading:

[1] Rabiner. L.R., and Gold, B., Theory and Application Of Digital Signal Processing, Englewood Cliffs, NJ: Prentice-Hall, 1975.

[2] Burrs, C.S., and Parks, T.W., DFT/FFT and Convolution Algorithms, New York: Wiley, 1985.

[3] Rodriguez, Jeffrey J., "An Improved FFT Digit-Reversal Algorithm," IEEE Transactions On Acoustics, Speech, And Signal Processing, Vol. 37, No. 8, Aug 1989.

4

TABLE 2: REQUIREMENTS FOR RADIX-2 FFT @25 MHz

N (FFT Length)	64 Point	256 Point	1024 Point
Code Space (locations)	603 + 0.75*N = 651	603 + 0.75*N = 795	603 + 0.75*N = 1371
Data Storage in Program Memory Space	2*N = 128	2*N = 512	2*N = 2048
Scratch RAM	49	49	49
Performance (Square Wave Input Data)	122384 (19.58 ms)	644416 (103.11 ms)	3192176 (510.75 ms)

FIGURE 3: 2-SRAM EXTERNAL MEMORY CONNECTION

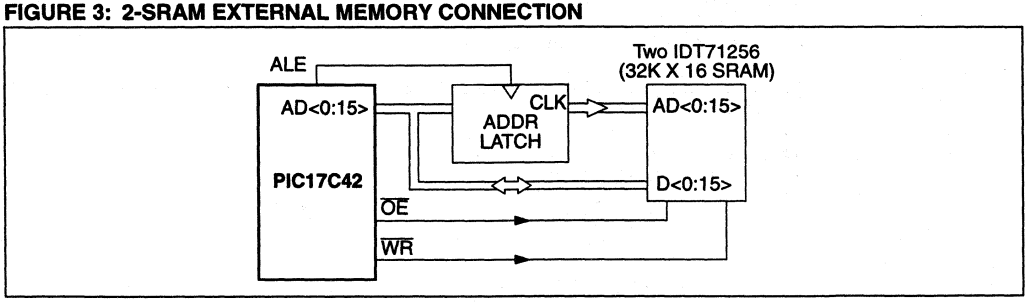
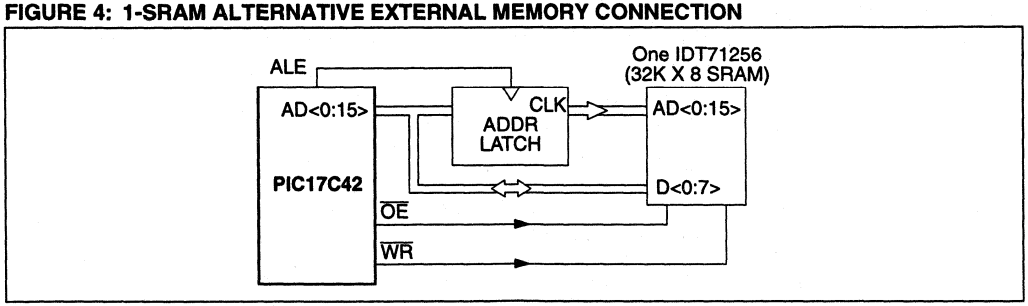


FIGURE 4: 1-SRAM ALTERNATIVE EXTERNAL MEMORY CONNECTION



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: FFT ALGORITHM

```
MPASM 01.40 Released            FFT.ASM   1-16-1997  14:54:45         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P=17C42, columns=120,WRAP, R=DEC
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;               A Cooley-Tukey Radix-2 DIF FFT
                     00005 ;
                     00006 ;       Radix-2 implementation
                     00007 ;       Decimation In Frequency
                     00008 ;       Single Butterfly
                     00009 ;       Table Lookup of Twiddle Factors
                     00010 ;       Complex Input & Complex Output
                     00011 ;
                     00012 ;       All data is assumed to be 16 bits and the intermediate
                     00013 ;       results are stored in 32 bits
                     00014 ;
                     00015 ;       Length Of FFT must be a Power Of 2
                     00016 ;       Max Length Possible is 2**15
                     00017 ;
                     00018 ;       The input/output complex data is organized as a single array
                     00019 ;       of real data followed by imaginary data
                     00020 ;               Data is stored in External Memory and is accessed by
                     00021 ;       TABLRD & TABLWT Instructions
                     00022 ;
                     00023 ;       Program:          FFT.ASM
                     00024 ;       Revision Date:
                     00025 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00026 ;
                     00027 ;*****************************************************************
                     00028 ;
                     00029         include <p17c42.inc>
                     00001         LIST
                     00002 ;P17C42.INC Standard Header File, Ver. 1.03 Microchip Technology, Inc.
                     00264         LIST
                     00030
                     00031         #define TRUE    1
                     00032         #define FALSE   0
                     00033
                     00034         #define LSB     0
                     00035         #define MSB     7
                     00036
                     00037         include <17c42.mac>
                     00945         LIST
                     00038
                     00039 ;*****************************************************************
                     00040 ;  RLC16AB
                     00041 ;
                     00042 ; DESCRIPTION:
                     00043 ;       16 bit rotate left A into B
                     00044 ;
                     00045 ; ARGUMENTS:
                     00046 ;       2*a => b
                     00047 ;
                     00048 ; TIMING (in cycles):
                     00049 ;       3
                     00050 ;
```


© 1997 Microchip Technology Inc.

```
00051 ;
00052
00053 RLC16AB MACRO   a,b
00054
00055    BCF     ALUSTA,C
00056    RLCF    a+BB0,W
00057    MOVWF   b+BB0
00058    RLCF    a+BB1,W
00059    MOVWF   b+BB1
00060
00061         ENDM
00062
00063 ;*****************************************************************
00064 ;       TBLADDR
00065 ;
00066 ; DESCRIPTION:
00067 ;       Load 16 bit table pointer with specified label
00068 ;
00069 ; TIMING (in cycles):
00070 ;       4
00071 ;
00072
00073 TBLADDR MACRO   label
00074
00075    MOVLW   LOW   label
00076    MOVWF   TBLPTRL
00077    MOVLW   HIGH   label
00078    MOVWF   TBLPTRH
00079
00080         ENDM
00081
00082 ;*****************************************************************
00083 ;  ADDLBL
00084 ;
00085 ; DESCRIPTION:
00086 ;  ADd A Label (16 bit constant) To A File Register (16 bit)
00087 ;
00088 ; TIMING (in cycles):
00089 ;       4
00090 ;
00091
00092 ADDLBL  MACRO   label,f
00093
00094    MOVLW   LOW   label
00095    ADDWF   f+BB0, F
00096    MOVLW   HIGH   label
00097    ADDWFC  f+BB1, F
00098
00099         ENDM
00100
00101 ;*****************************************************************
00102
00103 ;
00000100        00104 FftLen  .set    256             ; FFT Length
00000008        00105 Power   .set    8               ; (2**Power = FftLen)
000000EF        00106 DigitRevCount   .set    239     ; (FftLen-1) - (2**((Power+1)/2))
                00107
00000001        00108 SCALE_BUTTERFLY .set    TRUE    ; intermediate scaling performed
                00109
00000800        00110 EXT_RAM_START_ADDR      .set    0x0800  ; External Memory Data Storage
                00111                                         ; Start Addr
                00112 ;*****************************************************************
                00113 ;
                00114         CBLOCK  0
00000000        00115           BB0,BB1,BB2,BB3 ; RAM offset constants
                00116         ENDC
```

```
                    00117 ;
                    00118         CBLOCK  0x18
  00000018          00119           AARG,AARG1              ; 16 bit multiplier A
  0000001A          00120           BARG,BARG1              ; 16 bit multiplicand B
  0000001C          00121           DPX,DPX1,DPX2,DPX3      ; 32 bit multiplier result = A*B
                    00122         ENDC
                    00123 ;
                    00124         CBLOCK
  00000020          00125           ACC, ACC1, ACC2, ACC3   ; 32 bit accumulator for computations
                    00126         ENDC
                    00127 ;
                    00128         CBLOCK
  00000024          00129           count1,count11          ; N1
  00000026          00130           count2,count22          ; N2
  00000028          00131           QuartLen,QuartLen1      ; FftLen/4
                    00132         ENDC
                    00133 ;
                    00134         CBLOCK
  0000002A          00135           TF_Offset,TF_Offset1
  0000002C          00136           TF_Addr,TF_Addr1        ; twiddle factor address computations
  0000002E          00137           Cos,Cos1
  00000030          00138           Sin,Sin1
                    00139         ENDC
                    00140 ;
                    00141         CBLOCK
  00000032          00142           VarIloop,VarIloop1
  00000034          00143           VarJloop,VarJloop1
  00000036          00144           VarKloop
  00000037          00145           VarL,VarL1
                    00146         ENDC
                    00147 ;
                    00148         CBLOCK
  00000039          00149           Xi,Xi1
  0000003B          00150           Yi,Yi1
  0000003D          00151           X1,X11
  0000003F          00152           Y1,Y11
                    00153         ENDC
                    00154 ;
                    00155         CBLOCK
  00000041          00156           Xt,Xt1
  00000043          00157           Yt,Yt1
                    00158         ENDC
                    00159
                    00160         CBLOCK
  00000045          00161           temp,temp1,temp2,temp3
  00000049          00162           testCount,testCount1
  0000004B          00163           PulseCount, PulseCount1
                    00164         ENDC
                    00165 ;
                    00166 ;
                    00167 ;*****************************************************************
                    00168 ;               Test Program For FFT Subroutine
                    00169 ;*****************************************************************
                    00170 ;
0000                00171         ORG     0x0000
                    00172 ;
                    00173         include <square.asm>            ; Generate Test Vector Data
                    00001 ;*****************************************************************
                    00002 ;               Test Routine For FFT
                    00003 ; FFT Of Square Wave Pulse
                    00004 ;*****************************************************************
  00000008          00005 PulseWidthFactor        .set    8
                    00006 ;
0000                00007 testFft
                    00008   MOVK16  2*PulseWidthFactor,testCount
                        M
```


© 1997 Microchip Technology Inc.

```
0000 B010              M   MOVLW   (2*PulseWidthFactor) & 0xff
0001 0149              M   MOVWF   testCount+B0
0002 B000              M   MOVLW   ((2*PulseWidthFactor) >> 8)
0003 014A              M   MOVWF   testCount+B1
                       M
                   00009   CLR16   Yi
                       M
0004 293B              M   CLRF    Yi+B0, F
0005 293C              M   CLRF    Yi+B1, F
                       M
                   00010   TBLADDR ExtRamAddr      ; load table pointers with data start addr
                       M
0006 B000              M   MOVLW   LOW   ExtRamAddr
0007 010D              M   MOVWF   TBLPTRL
0008 B008              M   MOVLW   HIGH   ExtRamAddr
0009 010E              M   MOVWF   TBLPTRH
                       M
                   00011 ;
000A               00012 nextPulse
                   00013   MOVK    FftLen/PulseWidthFactor,PulseCount
                       M
000A B020              M   MOVLW   FftLen/PulseWidthFactor
000B 014B              M   MOVWF   PulseCount
                       M
                   00014   MOVK    FftLen/PulseWidthFactor,PulseCount1
                       M
000C B020              M   MOVLW   FftLen/PulseWidthFactor
000D 014C              M   MOVWF   PulseCount1
                       M
                   00015 ;
                   00016   MOVK16  0x3FFF,Xi
                       M
000E B0FF              M   MOVLW   (0x3FFF) & 0xff
000F 0139              M   MOVWF   Xi+B0
0010 B03F              M   MOVLW   ((0x3FFF) >> 8)
0011 013A              M   MOVWF   Xi+B1
                       M
0012               00017 LX1
0012 E034          00018   call    write
0013 174B          00019   decfsz  PulseCount, F
0014 C012          00020   goto    LX1
                   00021 ;
                   00022   CLR16   Xi
                       M
0015 2939              M   CLRF    Xi+B0, F
0016 293A              M   CLRF    Xi+B1, F
                       M
0017               00023 LX2
0017 E034          00024   call    write
0018 174C          00025   decfsz  PulseCount1, F
0019 C017          00026   goto    LX2
                   00027 ;
                   00028   DEC16   testCount
                       M
001A 290A              M   CLRF    WREG, F
001B 0749              M   DECF    testCount+B0, F
001C 034A              M   SUBWFB  testCount+B1, F
                       M
                   00029   TFSZ16  testCount
                       M
001D 6A49              M   MOVFP   testCount+B0,WREG
001E 084A              M   IORWF   testCount+B1,W
001F 330A              M   TSTFSZ  WREG
                       M
0020 C00A          00030   goto    nextPulse
                   00031
```

4

© 1997 Microchip Technology Inc.

```
                    00174 ;
0021 E039           00175    call    R2FFT           ; Compute Fourier Transform
0022 E116           00176    call    Unscramble      ; Digit Reverse the scrambled data
                    00177 ;
                    00178 ; Fourier Transform Completed
                    00179 ;
                    00180 ; capture data to PIC-MASTER Trace Buffer
                    00181   MOVK16  FftLen*2,temp
                        M
0023 B000               M   MOVLW   (FftLen*2) & 0xff
0024 0145               M   MOVWF   temp+B0
0025 B002               M   MOVLW   ((FftLen*2) >> 8)
0026 0146               M   MOVWF   temp+B1
                        M
                    00182   TBLADDR ExtRamAddr      ; load table pointers with data start addr
                        M
0027 B000               M   MOVLW   LOW   ExtRamAddr
0028 010D               M   MOVWF   TBLPTRL
0029 B008               M   MOVLW   HIGH   ExtRamAddr
002A 010E               M   MOVWF   TBLPTRH
                        M
002B                00183 capture
002B A930           00184   tablrd  0,1,Sin         ; table latch = mem(tblptr)
                    00185   DEC16   temp
                        M
002C 290A               M   CLRF    WREG, F
002D 0745               M   DECF    temp+B0, F
002E 0346               M   SUBWFB  temp+B1, F
                        M
                    00186   TFSZ16  temp
                        M
002F 6A45               M   MOVFP   temp+B0,WREG
0030 0846               M   IORWF   temp+B1,W
0031 330A               M   TSTFSZ  WREG
                        M
0032 C02B           00187   goto    capture
                    00188 ;
0033 C033           00189 self  goto    self
                    00190 ;
0034                00191 write
0034 A439           00192    tlwt    0,Xi
0035 AF3A           00193    tablwt  1,1,Xi+BB1      ; auto increment for Imag Data
0036 A43B           00194    tlwt    0,Yi
0037 AF3C           00195    tablwt  1,1,Yi+BB1
0038 0002           00196    return
                    00197 ;
                    00198 ;*****************************************************************
                    00199 ;                       RADIX-2 FFT
                    00200 ;
                    00201 ;       Decimation In Frequency
                    00202 ;
                    00203 ; Input Data should be unscrambled
                    00204 ; Output Data at the end is in scrambled form
                    00205 ; To obtain the unscrambled form, the digit reverse counter
                    00206 ; subroutine, "Unscramble" should be called (see the example)
                    00207 ;
                    00208 ;*****************************************************************
                    00209
0039                00210 R2FFT
                    00211    MOVK16  FftLen,count2       ; count2 = N
                        M
0039 B000               M    MOVLW   (FftLen) & 0xff
003A 0126               M    MOVWF   count2+B0
003B B001               M    MOVLW   ((FftLen) >> 8)
003C 0127               M    MOVWF   count2+B1
                        M
```


© 1997 Microchip Technology Inc.

```
                    00212   MOVK16  FftLen/4,QuartLen    ; QuartLen = FftLen/4
                        M
003D B040               M   MOVLW   (FftLen/4) & 0xff
003E 0128               M   MOVWF   QuartLen+B0
003F B000               M   MOVLW   ((FftLen/4) >> 8)
0040 0129               M   MOVWF   QuartLen+B1
                        M
0041 292B           00213   clrf    TF_Offset+BB1, F
                    00214   MOVK    1,TF_Offset         ; Init TF_Offset = 1
                        M
0042 B001               M   MOVLW   1
0043 012A               M   MOVWF   TF_Offset
                        M
                    00215   MOVK    Power,VarKloop      ; Kloop
                        M
0044 B008               M   MOVLW   Power
0045 0136               M   MOVWF   VarKloop
                        M
0046                00216 Kloop                         ; for K = 1 to Power-1
                    00217   MOV16   count2,count1       ; count1 = count2
                        M
0046 6A26               M   MOVFP   count2+B0,WREG      ; get byte of a into w
0047 0124               M   MOVWF   count1+B0           ; move to b(B0)
0048 6A27               M   MOVFP   count2+B1,WREG      ; get byte of a into w
0049 0125               M   MOVWF   count1+B1           ; move to b(B1)
                        M
                    00218   RRC16   count2              ; count2 = count2/2
                        M
004A 1A27               M   RLCF    count2+B1,W         ; move sign into carry bit
004B 1927               M   RRCF    count2+B1, F
004C 1926               M   RRCF    count2+B0, F
                        M
                    00219   CLR16   VarJloop            ; J = 0
                        M
004D 2934               M   CLRF    VarJloop+B0, F
004E 2935               M   CLRF    VarJloop+B1, F
                        M
                    00220   CLR16   TF_Addr             ; TF_Addr = 0
                        M
004F 292C               M   CLRF    TF_Addr+B0, F
0050 292D               M   CLRF    TF_Addr+B1, F
                        M
0051                00221 Jloop
                    00222 ;
                    00223 ; Read Twiddle factors from Sine/Cosine Table from Prog Mem
                    00224 ;
                    00225   MOVFP16 TF_Addr,TBLPTRL ; load sine table address to table pointers
                        M
0051 6D2C               M   MOVFP   TF_Addr+B0,TBLPTRL+B0   ; move A(B0) to B(B0)
0052 6E2D               M   MOVFP   TF_Addr+B1,TBLPTRL+B1   ; move A(B1) to B(B1)
                        M
                    00226   ADDLBL  SineTable,TBLPTRL       ; add table offset
                        M
0053 B094               M   MOVLW   LOW   SineTable
0054 0F0D               M   ADDWF   TBLPTRL+BB0, F
0055 B002               M   MOVLW   HIGH   SineTable
0056 110E               M   ADDWFC  TBLPTRL+BB1, F
                        M
                    00227
0057 A830           00228   tablrd  0,0,Sin             ; Read Sine Value from lookup table
0058 A030           00229   tlrd    0,Sin
0059 A231           00230   tlrd    1,Sin+BB1
                    00231   ADD16   QuartLen,TBLPTRL
                        M
005A 6A28               M   MOVFP   QuartLen+B0,WREG    ; get lowest byte of a into w
005B 0F0D               M   ADDWF   TBLPTRL+B0, F       ; add lowest byte of b, save in b(B0)
```


© 1997 Microchip Technology Inc.

```
005C 6A29              M    MOVFP   QuartLen+B1,WREG      ; get 2nd byte of a into w
005D 110E              M    ADDWFC  TBLPTRL+B1, F         ; add 2nd byte of b, save in b(B1)
                       M
005E A82E           00232   tablrd  0,0,Cos               ; Read Cosine Value from table
005F A02E           00233   tlrd    0,Cos
0060 A22F           00234   tlrd    1,Cos+BB1
                    00235 ;
                    00236   ADD16   TF_Offset,TF_Addr     ; TF_Addr = TF_Addr + TF_Offset
                       M
0061 6A2A              M    MOVFP   TF_Offset+B0,WREG     ; get lowest byte of a into w
0062 0F2C              M    ADDWF   TF_Addr+B0, F         ; add lowest byte of b, save in b(B0)
0063 6A2B              M    MOVFP   TF_Offset+B1,WREG     ; get 2nd byte of a into w
0064 112D              M    ADDWFC  TF_Addr+B1, F         ; add 2nd byte of b, save in b(B1)
                       M
                    00237 ;
                    00238   RLC16AB VarJloop,VarIloop  ; I=J*2 since Real followed by Imag Data
                       M
0065 8804              M    BCF     ALUSTA,C
0066 1A34              M    RLCF    VarJloop+BB0,W
0067 0132              M    MOVWF   VarIloop+BB0
0068 1A35              M    RLCF    VarJloop+BB1,W
0069 0133              M    MOVWF   VarIloop+BB1
                       M
                    00239 ;
006A                00240 Iloop
                    00241
                    00242   RLC16AB count2,VarL           ; VarL = count2*2
                       M
006A 8804              M    BCF     ALUSTA,C
006B 1A26              M    RLCF    count2+BB0,W
006C 0137              M    MOVWF   VarL+BB0
006D 1A27              M    RLCF    count2+BB1,W
006E 0138              M    MOVWF   VarL+BB1
                       M
                    00243   ADD16   VarIloop,VarL         ; VarL = (I+count2)*2
                       M
006F 6A32              M    MOVFP   VarIloop+B0,WREG      ; get lowest byte of a into w
0070 0F37              M    ADDWF   VarL+B0, F            ; add lowest byte of b, save in b(B0)
0071 6A33              M    MOVFP   VarIloop+B1,WREG      ; get 2nd byte of a into w
0072 1138              M    ADDWFC  VarL+B1, F            ; add 2nd byte of b, save in b(B1)
                       M
                    00244 ;
                    00245 ; Get Real & Imag Data from external RAMs (Program Memory)
                    00246 ; load table pointers with data start addr
                    00247 ;
                    00248   MOVFP16 VarL,TBLPTRL          ; read data(L)
                       M
0073 6D37              M    MOVFP   VarL+B0,TBLPTRL+B0    ; move A(B0) to B(B0)
0074 6E38              M    MOVFP   VarL+B1,TBLPTRL+B1    ; move A(B1) to B(B1)
                       M
                    00249   ADDLBL  ExtRamAddr,TBLPTRL    ; add data addr offset
                       M
0075 B000              M    MOVLW   LOW   ExtRamAddr
0076 0F0D              M    ADDWF   TBLPTRL+BB0, F
0077 B008              M    MOVLW   HIGH   ExtRamAddr
0078 110E              M    ADDWFC  TBLPTRL+BB1, F
                       M
                    00250
0079 A93D           00251   tablrd  0,1,Xl                ; auto increment for Imag Data
007A A03D           00252   tlrd    0,Xl
007B A23E           00253   tlrd    1,Xl+BB1              ; real data XL
007C A83F           00254   tablrd  0,0,Yl
007D A03F           00255   tlrd    0,Yl
007E A240           00256   tlrd    1,Yl+BB1              ; imag data YL
                    00257
                    00258   MOVFP16 VarIloop,TBLPTRL      ; read data(I)
```


© 1997 Microchip Technology Inc.

```
                    M
007F 6D32           M    MOVFP   VarIloop+B0,TBLPTRL+B0 ; move A(B0) to B(B0)
0080 6E33           M    MOVFP   VarIloop+B1,TBLPTRL+B1 ; move A(B1) to B(B1)
                    M
                00259    ADDLBL  ExtRamAddr,TBLPTRL    ; add data addr offset
                    M
0081 B000           M    MOVLW   LOW   ExtRamAddr
0082 0F0D           M    ADDWF   TBLPTRL+BB0, F
0083 B008           M    MOVLW   HIGH   ExtRamAddr
0084 110E           M    ADDWFC  TBLPTRL+BB1, F
                    M
                00260
0085 A939       00261    tablrd  0,1,Xi                ; auto increment for Imag Data
0086 A039       00262    tlrd    0,Xi
0087 A23A       00263    tlrd    1,Xi+BB1              ; real data XI
0088 A83B       00264    tablrd  0,0,Yi
0089 A03B       00265    tlrd    0,Yi
008A A23C       00266    tlrd    1,Yi+BB1              ; imag data YI
                00267 ;
                00268 ; Real & Imag Data is fetched
                00269 ; Compute Butterfly
                00270 ;
                00271    SUB16ACC       X1,Xi,Xt       ; Xt = Xi - Xl
                    M
008B 6A3D           M    MOVFP   X1+B0,WREG            ; get lowest byte of a into w
008C 0439           M    SUBWF   Xi+B0,W               ; sub lowest byte of b, save in b(B0)
008D 0141           M    MOVWF   Xt+B0
008E 6A3E           M    MOVFP   X1+B1,WREG            ; get 2nd byte of a into w
008F 023A           M    SUBWFB  Xi+B1,W               ; sub 2nd byte of b, save in b(B1)
0090 0142           M    MOVWF   Xt+B1
                    M
                00272    ADD16           X1,Xi         ; Xi = Xi + X1
                    M
0091 6A3D           M    MOVFP   X1+B0,WREG            ; get lowest byte of a into w
0092 0F39           M    ADDWF   Xi+B0, F              ; add lowest byte of b, save in b(B0)
0093 6A3E           M    MOVFP   X1+B1,WREG            ; get 2nd byte of a into w
0094 113A           M    ADDWFC  Xi+B1, F              ; add 2nd byte of b, save in b(B1)
                    M
                00273    SUB16ACC        Y1,Yi,Yt      ; Yt = Yi - Yl
                    M
0095 6A3F           M    MOVFP   Y1+B0,WREG            ; get lowest byte of a into w
0096 043B           M    SUBWF   Yi+B0,W               ; sub lowest byte of b, save in b(B0)
0097 0143           M    MOVWF   Yt+B0
0098 6A40           M    MOVFP   Y1+B1,WREG            ; get 2nd byte of a into w
0099 023C           M    SUBWFB  Yi+B1,W               ; sub 2nd byte of b, save in b(B1)
009A 0144           M    MOVWF   Yt+B1
                    M
                00274    ADD16           Y1,Yi         ; Yi = Yi + Yl
                    M
009B 6A3F           M    MOVFP   Y1+B0,WREG            ; get lowest byte of a into w
009C 0F3B           M    ADDWF   Yi+B0, F              ; add lowest byte of b, save in b(B0)
009D 6A40           M    MOVFP   Y1+B1,WREG            ; get 2nd byte of a into w
009E 113C           M    ADDWFC  Yi+B1, F              ; add 2nd byte of b, save in b(B1)
                    M
                00275 ;
                00276    #if SCALE_BUTTERFLY
                00277    RRC16   Xi
                    M
009F 1A3A           M    RLCF    Xi+B1,W               ; move sign into carry bit
00A0 193A           M    RRCF    Xi+B1, F
00A1 1939           M    RRCF    Xi+B0, F
                    M
                00278    RRC16   Yi
                    M
00A2 1A3C           M    RLCF    Yi+B1,W               ; move sign into carry bit
00A3 193C           M    RRCF    Yi+B1, F
```


© 1997 Microchip Technology Inc.

```
00A4 193B              M    RRCF    Yi+B0, F
                       M
                    00279   RRC16   Xt
                       M
00A5 1A42              M    RLCF    Xt+B1,W              ; move sign into carry bit
00A6 1942              M    RRCF    Xt+B1, F
00A7 1941              M    RRCF    Xt+B0, F
                       M
                    00280   RRC16   Yt
                       M
00A8 1A44              M    RLCF    Yt+B1,W              ; move sign into carry bit
00A9 1944              M    RRCF    Yt+B1, F
00AA 1943              M    RRCF    Yt+B0, F
                       M
                    00281   #endif
                    00282 ;
                    00283
                    00284   MOVFP16 Cos,AARG
                       M
00AB 782E              M    MOVFP   Cos+B0,AARG+B0       ; move A(B0) to B(B0)
00AC 792F              M    MOVFP   Cos+B1,AARG+B1       ; move A(B1) to B(B1)
                       M
                    00285   MOVFP16 Yt,BARG
                       M
00AD 7A43              M    MOVFP   Yt+B0,BARG+B0        ; move A(B0) to B(B0)
00AE 7B44              M    MOVFP   Yt+B1,BARG+B1        ; move A(B1) to B(B1)
                       M
00AF E182           00286   call    DblMult              ; COS*Yt
                    00287   MOVPF32 DPX,ACC
                       M
00B0 5C20              M    MOVPF   DPX+B0,ACC+B0        ; move A(B0) to B(B0)
00B1 5D21              M    MOVPF   DPX+B1,ACC+B1        ; move A(B1) to B(B1)
00B2 5E22              M    MOVPF   DPX+B2,ACC+B2        ; move A(B2) to B(B2)
00B3 5F23              M    MOVPF   DPX+B3,ACC+B3        ; move A(B3) to B(B3)
                       M
                    00288
                    00289   MOVFP16 Sin,AARG
                       M
00B4 7830              M    MOVFP   Sin+B0,AARG+B0       ; move A(B0) to B(B0)
00B5 7931              M    MOVFP   Sin+B1,AARG+B1       ; move A(B1) to B(B1)
                       M
                    00290   MOVFP16 Xt,BARG
                       M
00B6 7A41              M    MOVFP   Xt+B0,BARG+B0        ; move A(B0) to B(B0)
00B7 7B42              M    MOVFP   Xt+B1,BARG+B1        ; move A(B1) to B(B1)
                       M
00B8 E182           00291   Call    DblMult              ; SIN*Xt, Scale if necessary
                    00292
                    00293   ADD32   ACC,DPX
                       M
00B9 6A20              M    MOVFP   ACC+B0,WREG          ; get lowest byte of a into w
00BA 0F1C              M    ADDWF   DPX+B0, F            ; add lowest byte of b, save in b(B0)
00BB 6A21              M    MOVFP   ACC+B1,WREG          ; get 2nd byte of a into w
00BC 111D              M    ADDWFC  DPX+B1, F            ; add 2nd byte of b, save in b(B1)
00BD 6A22              M    MOVFP   ACC+B2,WREG          ; get 3rd byte of a into w
00BE 111E              M    ADDWFC  DPX+B2, F            ; add 3rd byte of b, save in b(B2)
00BF 6A23              M    MOVFP   ACC+B3,WREG          ; get 4th byte of a into w
00C0 111F              M    ADDWFC  DPX+B3, F            ; add 4th byte of b, save in b(B3)
                       M
                    00294   MOVPF16 DPX+BB2,Yl     ; Yl = COS*Yt + SIN*Xt, Scale if necessary
                       M
00C1 5E3F              M    MOVPF   DPX+BB2+B0,Yl+B0     ; move A(B0) to B(B0)
00C2 5F40              M    MOVPF   DPX+BB2+B1,Yl+B1     ; move A(B1) to B(B1)
                       M
                    00295 ;
                    00296   MOVFP16 Yt,BARG              ; AARG = SIN, BARG = Yt
```


© 1997 Microchip Technology Inc.

```
                        M
00C3 7A43               M     MOVFP   Yt+B0,BARG+B0          ; move A(B0) to B(B0)
00C4 7B44               M     MOVFP   Yt+B1,BARG+B1          ; move A(B1) to B(B1)
                        M
00C5 E182           00297     Call    DblMult                ; SIN*Yt
                    00298     MOVPF32 DPX,ACC
                        M
00C6 5C20               M     MOVPF   DPX+B0,ACC+B0          ; move A(B0) to B(B0)
00C7 5D21               M     MOVPF   DPX+B1,ACC+B1          ; move A(B1) to B(B1)
00C8 5E22               M     MOVPF   DPX+B2,ACC+B2          ; move A(B2) to B(B2)
00C9 5F23               M     MOVPF   DPX+B3,ACC+B3          ; move A(B3) to B(B3)
                        M
                    00299
                    00300     MOVFP16 Cos,AARG
                        M
00CA 782E               M     MOVFP   Cos+B0,AARG+B0         ; move A(B0) to B(B0)
00CB 792F               M     MOVFP   Cos+B1,AARG+B1         ; move A(B1) to B(B1)
                        M
                    00301     MOVFP16 Xt,BARG
                        M
00CC 7A41               M     MOVFP   Xt+B0,BARG+B0          ; move A(B0) to B(B0)
00CD 7B42               M     MOVFP   Xt+B1,BARG+B1          ; move A(B1) to B(B1)
                        M
00CE E182           00302     Call    DblMult                ; COS*Xt, Scale if necessary
                    00303
                    00304     SUB32   ACC,DPX                ; DPX = COS*Xt - SIN*Yt
                        M
00CF 6A20               M     MOVFP   ACC+B0,WREG            ; get lowest byte of a into w
00D0 051C               M     SUBWF   DPX+B0, F              ; sub lowest byte of b, save in b(B0)
00D1 6A21               M     MOVFP   ACC+B1,WREG            ; get 2nd byte of a into w
00D2 031D               M     SUBWFB  DPX+B1, F              ; sub 2nd byte of b, save in b(B1)
00D3 6A22               M     MOVFP   ACC+B2,WREG            ; get 3rd byte of a into w
00D4 031E               M     SUBWFB  DPX+B2, F              ; sub 3rd byte of b, save in b(B2)
00D5 6A23               M     MOVFP   ACC+B3,WREG            ; get 4th byte of a into w
00D6 031F               M     SUBWFB  DPX+B3, F              ; sub 4th byte of b, save in b(B3)
                        M
                    00305     MOVPF16 DPX+BB2,X1      ; X1 = COS*Xt - SIN*Yt, Scale if necessary
                        M
00D7 5E3D               M     MOVPF   DPX+BB2+B0,X1+B0       ; move A(B0) to B(B0)
00D8 5F3E               M     MOVPF   DPX+BB2+B1,X1+B1       ; move A(B1) to B(B1)
                        M
                    00306 ;
                    00307 ;
                    00308 ; Store results of butterfly
                    00309 ;
                    00310     DEC16   TBLPTRL        ; table pointer already loaded with I addr
                        M
00D9 290A               M     CLRF    WREG, F
00DA 070D               M     DECF    TBLPTRL+B0, F
00DB 030E               M     SUBWFB  TBLPTRL+B1, F
                        M
                    00311
00DC A439           00312     tlwt    0,Xi
00DD AF3A           00313     tablwt  1,1,Xi+BB1             ; auto increment for Imag Data
00DE A43B           00314     tlwt    0,Yi
00DF AE3C           00315     tablwt  1,0,Yi+BB1             ; Xi & Yi stored
                    00316
                    00317     MOVFP16 VarL,TBLPTRL           ; read data(L)
                        M
00E0 6D37               M     MOVFP   VarL+B0,TBLPTRL+B0     ; move A(B0) to B(B0)
00E1 6E38               M     MOVFP   VarL+B1,TBLPTRL+B1     ; move A(B1) to B(B1)
                        M
                    00318     ADDLBL  ExtRamAddr,TBLPTRL     ; add data addr offset
                        M
00E2 B000               M     MOVLW   LOW   ExtRamAddr
00E3 0F0D               M     ADDWF   TBLPTRL+BB0, F
```

```
00E4 B008              M    MOVLW   HIGH   ExtRamAddr
00E5 110E              M    ADDWFC  TBLPTRL+BB1, F
                       M
                    00319
00E6 A43D           00320    tlwt    0,Xl
00E7 AF3E           00321    tablwt  1,1,Xl+BB1            ; auto increment for Imag Data
00E8 A43F           00322    tlwt    0,Yl
00E9 AE40           00323    tablwt  1,0,Yl+BB1            ; X(L) & Y(L) stored
                    00324 ;
                    00325 ; Increment for next Iloop
                    00326 ;
                    00327    RLC16AB count1,temp           ; temp = count1*2
                       M
00EA 8804              M    BCF     ALUSTA,C
00EB 1A24              M    RLCF    count1+BB0,W
00EC 0145              M    MOVWF   temp+BB0
00ED 1A25              M    RLCF    count1+BB1,W
00EE 0146              M    MOVWF   temp+BB1
                       M
                    00328    ADD16   temp,VarIloop         ; I = I + temp
                       M
00EF 6A45              M    MOVFP   temp+B0,WREG          ; get lowest byte of a into w
00F0 0F32              M    ADDWF   VarIloop+B0, F        ; add lowest byte of b, save in b(B0)
00F1 6A46              M    MOVFP   temp+B1,WREG          ; get 2nd byte of a into w
00F2 1133              M    ADDWFC  VarIloop+B1, F        ; add 2nd byte of b, save in b(B1)
                       M
                    00329 ;
                    00330
                    00331    MOVK16  (FftLen*2),temp
                       M
00F3 B000              M    MOVLW   ((FftLen*2)) & 0xff
00F4 0145              M    MOVWF   temp+B0
00F5 B002              M    MOVLW   (((FftLen*2)) >> 8)
00F6 0146              M    MOVWF   temp+B1
                       M
                    00332    SUB16   VarIloop,temp         ; temp = 2*FftLen - I
                       M
00F7 6A32              M    MOVFP   VarIloop+B0,WREG      ; get lowest byte of a into w
00F8 0545              M    SUBWF   temp+B0, F            ; sub lowest byte of b, save in b(B0)
00F9 6A33              M    MOVFP   VarIloop+B1,WREG      ; get 2nd byte of a into w
00FA 0346              M    SUBWFB  temp+B1, F            ; sub 2nd byte of b, save in b(B1)
                       M
                    00333    DEC16   temp
                       M
00FB 290A              M    CLRF    WREG, F
00FC 0745              M    DECF    temp+B0, F
00FD 0346              M    SUBWFB  temp+B1, F
                       M
00FE 9746           00334    Btfss   temp+BB1,MSB
00FF C06A           00335    Goto    Iloop                 ; while I < 2*FftLen
                    00336 ;
                    00337 ; I Loop end
                    00338 ;
                    00339 ; increment for next J Loop
                    00340 ;
                    00341
                    00342    INC16   VarJloop              ; J = J + 1
                       M
0100 290A              M    CLRF    WREG, F
0101 1534              M         INCF    (VarJloop)+B0, F
0102 1135              M    ADDWFC  (VarJloop)+B1, F
                       M
                    00343
                    00344    MOV16   count2,temp
                       M
0103 6A26              M    MOVFP   count2+B0,WREG        ; get byte of a into w
```


© 1997 Microchip Technology Inc.

```
0104 0145              M    MOVWF   temp+B0                 ; move to b(B0)
0105 6A27              M    MOVFP   count2+B1,WREG          ; get byte of a into w
0106 0146              M    MOVWF   temp+B1                 ; move to b(B1)
                       M
                    00345   SUB16   VarJloop,temp           ; temp = count2 - J
                       M
0107 6A34              M    MOVFP   VarJloop+B0,WREG        ; get lowest byte of a into w
0108 0545              M    SUBWF   temp+B0, F              ; sub lowest byte of b, save in b(B0)
0109 6A35              M    MOVFP   VarJloop+B1,WREG        ; get 2nd byte of a into w
010A 0346              M    SUBWFB  temp+B1, F              ; sub 2nd byte of b, save in b(B1)
                       M
                    00346   DEC16   temp
                       M
010B 290A              M    CLRF    WREG, F
010C 0745              M    DECF    temp+B0, F
010D 0346              M    SUBWFB  temp+B1, F
                       M
010E 9746           00347   Btfss   temp+BB1,MSB
010F C051           00348   Goto    Jloop                   ; while J < count2
                    00349 ;
                    00350 ; J Loop end
                    00351 ;
                    00352 ; increment for next K Loop
                    00353 ;
                    00354   RLC16   TF_Offset               ; TF_Offset = 2 * TF_Offset
0110 8804              M    BCF     ALUSTA,C
0111 1B2A              M    RLCF    TF_Offset+B0, F
0112 1B2B              M    RLCF    TF_Offset+B1, F
                       M
0113 1736           00355   decfsz  VarKloop, F
0114 C046           00356   Goto    Kloop                   ; while K < Power
                    00357 ;
0115 0002           00358   return                          ; FFT complete
                    00359 ;
                    00360 ; K Loop End
                    00361 ; FFT Computation Over with data scrambled
                    00362 ; Descramble the data using "Unscramble" Routine
                    00363 ;
                    00364 ;*****************************************************************
                    00365 ;       Unscramble Data Order Sequence
                    00366 ;       A digit reverse counter
                    00367 ;*****************************************************************
                    00368
                    00369         include <reverse.asm>
                    00001 ;*****************************************************************
                    00002 ;                       A digit reverse counter
                    00003 ;
                    00004 ;       Unscramble Data Order Sequence Of Radix-2 FFT
                    00005 ;       Length (must be a power of 2) is limited only by
                    00006 ;       the amount of External RAM available and must be
                    00007 ;       a number less than 2**15
                    00008 ;
                    00009 ;*****************************************************************
                    00010
0116                00011 Unscramble
                    00012
                    00013   CLR16   VarJloop        ; J = 0
                       M
0116 2934              M    CLRF    VarJloop+B0, F
0117 2935              M    CLRF    VarJloop+B1, F
                       M
0118 2933           00014   ClRF    VarIloop+B1, F
                    00015   MOVK    1,VarIloop      ; I = 1
                       M
0119 B001              M    MOVLW   1
011A 0132              M    MOVWF   VarIloop
```

```
                        M
011B                00016 nextI
                    00017     MOVK16   FftLen/2,VarKloop
                        M
011B B080               M     MOVLW    (FftLen/2) & 0xff
011C 0136               M     MOVWF    VarKloop+B0
011D B000               M     MOVLW    ((FftLen/2) >> 8)
011E 0137               M     MOVWF    VarKloop+B1
                        M
011F C127           00018     Goto     testK
0120                00019 KlessJ
                    00020     SUB16    VarKloop,VarJloop      ; J = J - K
                        M
0120 6A36               M     MOVFP    VarKloop+B0,WREG       ; get lowest byte of a into w
0121 0534               M     SUBWF    VarJloop+B0, F         ; sub lowest byte of b, save in b(B0)
0122 6A37               M     MOVFP    VarKloop+B1,WREG       ; get 2nd byte of a into w
0123 0335               M     SUBWFB   VarJloop+B1, F         ; sub 2nd byte of b, save in b(B1)
                        M
                    00021     RRC16    VarKloop               ; K = K/2
                        M
0124 1A37               M     RLCF     VarKloop+B1,W          ; move sign into carry bit
0125 1937               M     RRCF     VarKloop+B1, F
0126 1936               M     RRCF     VarKloop+B0, F
                        M
0127                00022 testK
                    00023     MOV16    VarJloop,temp
                        M
0127 6A34               M     MOVFP    VarJloop+B0,WREG       ; get byte of a into w
0128 0145               M     MOVWF    temp+B0                ; move to b(B0)
0129 6A35               M     MOVFP    VarJloop+B1,WREG       ; get byte of a into w
012A 0146               M     MOVWF    temp+B1                ; move to b(B1)
                        M
                    00024     SUB16    VarKloop,temp          ; temp = J - K
                        M
012B 6A36               M     MOVFP    VarKloop+B0,WREG       ; get lowest byte of a into w
012C 0545               M     SUBWF    temp+B0, F             ; sub lowest byte of b, save in b(B0)
012D 6A37               M     MOVFP    VarKloop+B1,WREG       ; get 2nd byte of a into w
012E 0346               M     SUBWFB   temp+B1, F             ; sub 2nd byte of b, save in b(B1)
                        M
012F 9746           00025     Btfss    temp+B1,MSB
0130 C120           00026     goto     KlessJ                 ; while K < J
                    00027
                    00028     ADD16    VarKloop,VarJloop      ; J = J + K
                        M
0131 6A36               M     MOVFP    VarKloop+B0,WREG       ; get lowest byte of a into w
0132 0F34               M     ADDWF    VarJloop+B0, F         ; add lowest byte of b, save in b(B0)
0133 6A37               M     MOVFP    VarKloop+B1,WREG       ; get 2nd byte of a into w
0134 1135               M     ADDWFC   VarJloop+B1, F         ; add 2nd byte of b, save in b(B1)
                        M
                    00029 ;
                    00030 ; if (i < j) then swap data(i) & data(j)
                    00031 ;
                    00032     MOV16    VarJloop,temp
                        M
0135 6A34               M     MOVFP    VarJloop+B0,WREG       ; get byte of a into w
0136 0145               M     MOVWF    temp+B0                ; move to b(B0)
0137 6A35               M     MOVFP    VarJloop+B1,WREG       ; get byte of a into w
0138 0146               M     MOVWF    temp+B1                ; move to b(B1)
                        M
                    00033     SUB16    VarIloop,temp          ; temp = J - I
                        M
0139 6A32               M     MOVFP    VarIloop+B0,WREG       ; get lowest byte of a into w
013A 0545               M     SUBWF    temp+B0, F             ; sub lowest byte of b, save in b(B0)
013B 6A33               M     MOVFP    VarIloop+B1,WREG       ; get 2nd byte of a into w
013C 0346               M     SUBWFB   temp+B1, F             ; sub 2nd byte of b, save in b(B1)
                        M
```


© 1997 Microchip Technology Inc.

```
                  00034     DEC16   temp
                      M
013D 290A             M     CLRF    WREG, F
013E 0745             M     DECF    temp+B0, F
013F 0346             M     SUBWFB  temp+B1, F
                      M
0140 9F46         00035     Btfsc   temp+B1,MSB
0141 C174         00036     Goto    incI
                  00037 ;
                  00038 ; swap data
                  00039 ; read data(i)
                  00040
                  00041     RLC16AB VarIloop,TBLPTRL      ; add twice the addr, since Real Data
                      M
0142 8804             M     BCF     ALUSTA,C
0143 1A32             M     RLCF    VarIloop+BB0,W
0144 010D             M     MOVWF   TBLPTRL+BB0
0145 1A33             M     RLCF    VarIloop+BB1,W
0146 010E             M     MOVWF   TBLPTRL+BB1
                      M
                  00042     ADDLBL  ExtRamAddr,TBLPTRL    ; is followed by Imag Data
                      M
0147 B000             M     MOVLW   LOW   ExtRamAddr
0148 0F0D             M     ADDWF   TBLPTRL+BB0, F
0149 B008             M     MOVLW   HIGH   ExtRamAddr
014A 110E             M     ADDWFC  TBLPTRL+BB1, F
                      M
                  00043
014B A939         00044     tablrd  0,1,Xi                ; auto increment for Imag Data
014C A039         00045     tlrd    0,Xi
014D A23A         00046     tlrd    1,Xi+B1               ; real data XI
014E A83B         00047     tablrd  0,0,Yi
014F A03B         00048     tlrd    0,Yi
0150 A23C         00049     tlrd    1,Yi+B1               ; imag data YI
                  00050 ;
                  00051 ; read data(j)
                  00052 ;
                  00053     RLC16AB VarJloop,TBLPTRL      ; add twice the addr, since Real Data
                      M
0151 8804             M     BCF     ALUSTA,C
0152 1A34             M     RLCF    VarJloop+BB0,W
0153 010D             M     MOVWF   TBLPTRL+BB0
0154 1A35             M     RLCF    VarJloop+BB1,W
0155 010E             M     MOVWF   TBLPTRL+BB1
                      M
                  00054     ADDLBL  ExtRamAddr,TBLPTRL    ; is followed by Imag Data
                      M
0156 B000             M     MOVLW   LOW   ExtRamAddr
0157 0F0D             M     ADDWF   TBLPTRL+BB0, F
0158 B008             M     MOVLW   HIGH   ExtRamAddr
0159 110E             M     ADDWFC  TBLPTRL+BB1, F
                      M
                  00055
015A A93D         00056     tablrd  0,1,Xl                ; auto increment for Imag Data
015B A03D         00057     tlrd    0,Xl
015C A23E         00058     tlrd    1,Xl+B1               ; real data XL
015D A83F         00059     tablrd  0,0,Yl
015E A03F         00060     tlrd    0,Yl
015F A240         00061     tlrd    1,Yl+B1               ; imag data YL
                  00062 ;
                  00063 ; Interchange data(I) & data(J)
                  00064 ;
                  00065 ; J addr already loaded into table pointers, by auto incremented
                  00066 ;
                  00067     DEC16   TBLPTRL
                      M
```


© 1997 Microchip Technology Inc.

```
0160 290A              M    CLRF    WREG, F
0161 070D              M    DECF    TBLPTRL+B0, F
0162 030E              M    SUBWFB  TBLPTRL+B1, F
                       M
                    00068
0163 A439           00069    tlwt    0,Xi
0164 AF3A           00070    tablwt  1,1,Xi+B1            ; auto increment for Imag Data
0165 A43B           00071    tlwt    0,Yi
0166 AE3C           00072    tablwt  1,0,Yi+B1            ; X(I) & Y(I) stored
                    00073
                    00074    RLC16AB VarIloop,TBLPTRL     ; add twice the addr, since Real Data
                       M
0167 8804              M    BCF     ALUSTA,C
0168 1A32              M    RLCF    VarIloop+BB0,W
0169 010D              M    MOVWF   TBLPTRL+BB0
016A 1A33              M    RLCF    VarIloop+BB1,W
016B 010E              M    MOVWF   TBLPTRL+BB1
                       M
                    00075    ADDLBL  ExtRamAddr,TBLPTRL   ; is followed by Imag Data
                       M
016C B000              M    MOVLW   LOW   ExtRamAddr
016D 0F0D              M    ADDWF   TBLPTRL+BB0, F
016E B008              M    MOVLW   HIGH   ExtRamAddr
016F 110E              M    ADDWFC  TBLPTRL+BB1, F
                       M
                    00076
0170 A43D           00077    tlwt    0,Xl
0171 AF3E           00078    tablwt  1,1,Xl+B1            ; auto increment for Imag Data
0172 A43F           00079    tlwt    0,Yl
0173 AE40           00080    tablwt  1,0,Yl+B1            ; X(L) & Y(L) stored
                    00081 ;
                    00082 ; increment I
                    00083 ;
0174                00084 incI
                    00085   INC16   VarIloop
                       M
0174 290A              M    CLRF    WREG, F
0175 1532              M    INCF    (VarIloop)+B0, F
0176 1133              M    ADDWFC  (VarIloop)+B1, F
                       M
                    00086
                    00087    MOVK16  DigitRevCount,temp
                       M
0177 B0EF              M    MOVLW   (DigitRevCount) & 0xff
0178 0145              M    MOVWF   temp+B0
0179 B000              M    MOVLW   ((DigitRevCount) >> 8)
017A 0146              M    MOVWF   temp+B1
                       M
                    00088    SUB16   VarIloop,temp        ; temp = DigitRevCount - I
                       M
017B 6A32              M    MOVFP   VarIloop+B0,WREG     ; get lowest byte of a into w
017C 0545              M    SUBWF   temp+B0, F           ; sub lowest byte of b, save in b(B0)
017D 6A33              M    MOVFP   VarIloop+B1,WREG     ; get 2nd byte of a into w
017E 0346              M    SUBWFB  temp+B1, F           ; sub 2nd byte of b, save in b(B1)
                       M
017F 9746           00089    Btfss   temp+B1,MSB
0180 C11B           00090    Goto    nextI                ; while i < DigitRevCount
0181 0002           00091    return
                    00092 ;
                    00093 ; End digit reverse counter
                    00094 ;
                    00095 ;*****************************************************************
                    00096
                    00370
                    00371 ;*****************************************************************
                    00372 ;       Include Double Precision Multiplication Routine
```


© 1997 Microchip Technology Inc.

```
                    00373 ;*****************************************************************
  00000001          00374 SIGNED  equ     TRUE
                    00375
                    00376         include <17c42mpy.mac>
                    00178         LIST
                    00377
                    00378 ;
                    00379 ;*****************************************************************
                    00380 ;               Sine-Cosine Tables
                    00381 ;*****************************************************************
                    00382 ;
                    00383         include <fft256.tbl>
                    00001 ;
                    00002 ;               256 Point FFT Sine Table
                    00003 ; coefficient table (size of table is 3n/4).
                    00004 ;
0294                00005 SineTable
                    00006 ;
0294 0000           00007         data          0
0295 0324           00008         data          804
0296 0648           00009         data          1608
0297 096A           00010         data          2410
0298 0C8C           00011         data          3212
0299 0FAB           00012         data          4011
029A 12C8           00013         data          4808
029B 15E2           00014         data          5602
029C 18F9           00015         data          6393
029D 1C0B           00016         data          7179
029E 1F1A           00017         data          7962
029F 2223           00018         data          8739
02A0 2528           00019         data          9512
02A1 2826           00020         data          10278
02A2 2B1F           00021         data          11039
02A3 2E11           00022         data          11793
02A4 30FB           00023         data          12539
02A5 33DF           00024         data          13279
02A6 36BA           00025         data          14010
02A7 398C           00026         data          14732
02A8 3C56           00027         data          15446
02A9 3F17           00028         data          16151
02AA 41CE           00029         data          16846
02AB 447A           00030         data          17530
02AC 471C           00031         data          18204
02AD 49B4           00032         data          18868
02AE 4C3F           00033         data          19519
02AF 4EBF           00034         data          20159
02B0 5133           00035         data          20787
02B1 539B           00036         data          21403
02B2 55F5           00037         data          22005
02B3 5842           00038         data          22594
02B4 5A82           00039         data          23170
02B5 5CB3           00040         data          23731
02B6 5ED7           00041         data          24279
02B7 60EB           00042         data          24811
02B8 62F1           00043         data          25329
02B9 64E8           00044         data          25832
02BA 66CF           00045         data          26319
02BB 68A6           00046         data          26790
02BC 6A6D           00047         data          27245
02BD 6C23           00048         data          27683
02BE 6DC9           00049         data          28105
02BF 6F5E           00050         data          28510
02C0 70E2           00051         data          28898
02C1 7254           00052         data          29268
02C2 73B5           00053         data          29621
02C3 7504           00054         data          29956
```


© 1997 Microchip Technology Inc.

```
02C4 7641          00055        data          30273
02C5 776B          00056        data          30571
02C6 7884          00057        data          30852
02C7 7989          00058        data          31113
02C8 7A7C          00059        data          31356
02C9 7B5C          00060        data          31580
02CA 7C29          00061        data          31785
02CB 7CE3          00062        data          31971
02CC 7D89          00063        data          32137
02CD 7E1D          00064        data          32285
02CE 7E9C          00065        data          32412
02CF 7F09          00066        data          32521
02D0 7F61          00067        data          32609
02D1 7FA6          00068        data          32678
02D2 7FD8          00069        data          32728
02D3 7FF5          00070        data          32757
                   00071 ;
02D4               00072 CosTable
                   00073 ;
02D4 7FFF          00074        data          32767
02D5 7FF5          00075        data          32757
02D6 7FD8          00076        data          32728
02D7 7FA6          00077        data          32678
02D8 7F61          00078        data          32609
02D9 7F09          00079        data          32521
02DA 7E9C          00080        data          32412
02DB 7E1D          00081        data          32285
02DC 7D89          00082        data          32137
02DD 7CE3          00083        data          31971
02DE 7C29          00084        data          31785
02DF 7B5C          00085        data          31580
02E0 7A7C          00086        data          31356
02E1 7989          00087        data          31113
02E2 7884          00088        data          30852
02E3 776B          00089        data          30571
02E4 7641          00090        data          30273
02E5 7504          00091        data          29956
02E6 73B5          00092        data          29621
02E7 7254          00093        data          29268
02E8 70E2          00094        data          28898
02E9 6F5E          00095        data          28510
02EA 6DC9          00096        data          28105
02EB 6C23          00097        data          27683
02EC 6A6D          00098        data          27245
02ED 68A6          00099        data          26790
02EE 66CF          00100        data          26319
02EF 64E8          00101        data          25832
02F0 62F1          00102        data          25329
02F1 60EB          00103        data          24811
02F2 5ED7          00104        data          24279
02F3 5CB3          00105        data          23731
02F4 5A82          00106        data          23170
02F5 5842          00107        data          22594
02F6 55F5          00108        data          22005
02F7 539B          00109        data          21403
02F8 5133          00110        data          20787
02F9 4EBF          00111        data          20159
02FA 4C3F          00112        data          19519
02FB 49B4          00113        data          18868
02FC 471C          00114        data          18204
02FD 447A          00115        data          17530
02FE 41CE          00116        data          16846
02FF 3F17          00117        data          16151
0300 3C56          00118        data          15446
0301 398C          00119        data          14732
0302 36BA          00120        data          14010
```


© 1997 Microchip Technology Inc.

```
0303 33DF       00121       data        13279
0304 30FB       00122       data        12539
0305 2E11       00123       data        11793
0306 2B1F       00124       data        11039
0307 2826       00125       data        10278
0308 2528       00126       data        9512
0309 2223       00127       data        8739
030A 1F1A       00128       data        7962
030B 1C0B       00129       data        7179
030C 18F9       00130       data        6393
030D 15E2       00131       data        5602
030E 12C8       00132       data        4808
030F 0FAB       00133       data        4011
0310 0C8C       00134       data        3212
0311 096A       00135       data        2410
0312 0648       00136       data        1608
0313 0324       00137       data        804
0314 0000       00138       data        0
0315 FCDC       00139       data        -804
0316 F9B8       00140       data        -1608
0317 F696       00141       data        -2410
0318 F374       00142       data        -3212
0319 F055       00143       data        -4011
031A ED38       00144       data        -4808
031B EA1E       00145       data        -5602
031C E707       00146       data        -6393
031D E3F5       00147       data        -7179
031E E0E6       00148       data        -7962
031F DDDD       00149       data        -8739
0320 DAD8       00150       data        -9512
0321 D7DA       00151       data        -10278
0322 D4E1       00152       data        -11039
0323 D1EF       00153       data        -11793
0324 CF05       00154       data        -12539
0325 CC21       00155       data        -13279
0326 C946       00156       data        -14010
0327 C674       00157       data        -14732
0328 C3AA       00158       data        -15446
0329 C0E9       00159       data        -16151
032A BE32       00160       data        -16846
032B BB86       00161       data        -17530
032C B8E4       00162       data        -18204
032D B64C       00163       data        -18868
032E B3C1       00164       data        -19519
032F B141       00165       data        -20159
0330 AECD       00166       data        -20787
0331 AC65       00167       data        -21403
0332 AA0B       00168       data        -22005
0333 A7BE       00169       data        -22594
0334 A57E       00170       data        -23170
0335 A34D       00171       data        -23731
0336 A129       00172       data        -24279
0337 9F15       00173       data        -24811
0338 9D0F       00174       data        -25329
0339 9B18       00175       data        -25832
033A 9931       00176       data        -26319
033B 975A       00177       data        -26790
033C 9593       00178       data        -27245
033D 93DD       00179       data        -27683
033E 9237       00180       data        -28105
033F 90A2       00181       data        -28510
0340 8F1E       00182       data        -28898
0341 8DAC       00183       data        -29268
0342 8C4B       00184       data        -29621
0343 8AFC       00185       data        -29956
0344 89BF       00186       data        -30273
```


© 1997 Microchip Technology Inc.

```
0345 8895           00187        data          -30571
0346 877C           00188        data          -30852
0347 8677           00189        data          -31113
0348 8584           00190        data          -31356
0349 84A4           00191        data          -31580
034A 83D7           00192        data          -31785
034B 831D           00193        data          -31971
034C 8277           00194        data          -32137
034D 81E3           00195        data          -32285
034E 8164           00196        data          -32412
034F 80F7           00197        data          -32521
0350 809F           00198        data          -32609
0351 805A           00199        data          -32678
0352 8028           00200        data          -32728
0353 800B           00201        data          -32757
                    00202 ;
                    00203 ;***********************************************
                    00204
                    00384 ;
                    00385 ;*****************************************************************
                    00386 ;       FFT Input/Output Data Stored In External RAM
                    00387 ; Operate Processor In Extended Microcontroller Mode
                    00388 ; External Data Starts at Address 0x0800, with 2 bytes of
                    00389 ; Real Data followed by 2 bytes of Imaginary Data.
                    00390 ;*****************************************************************
0800                00391         ORG     EXT_RAM_START_ADDR
                    00392 ;
0800                00393 ExtRamAddr
                    00394 ;
                    00395         END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0280 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
02C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0300 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0340 : XXXXXXXXXXXXXXXX XXXX------------ ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   852


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN543

Tone Generation

Author: *Amar Palacherla*
Microchip Technology Inc.

INTRODUCTION

A general purpose resonator routine is implemented using a PIC17C42. This routine is used to generate multiple tones. A tone signal is normally generated using extensive table lookup schemes. When a multiple tone signal is desired, each tone must have its own lookup table, thus requiring a large amount of storage space, especially when various frequencies are to be generated. This application note implements tone generation using recursive techniques. The algorithm for a resonator is developed and implemented using PIC17C42.

THEORY

Generation of a single tone basically implies generating samples of a sine/cosine wave. The Z-Transform of a sine wave is given as follows:

$$Z\{sin(wt)\} = \frac{Y(z)}{X(z)} = \frac{z^{*}sin(wT)}{z^{2} - 2^{*}z^{*}cos(wT) + 1}$$

The impulse response of the above transform (i.e., for X(z) = 1) will generate a sine wave of frequency w sampled at a rate of T (= 1/fs). Thus the above equation is translated to:

$$Y(z) = \frac{z^{-1^{*}}sin(wT)}{1 - 2^{*}z^{-1^{*}}cos(wT) + z^{2}}$$

The above equation can be rewritten in a difference equation form as follows:

$$y(n) - 2y(n-1)\cos(wT) + y(n-2) = x(n-1)sin(wT)$$

Rearranging the above equation and setting, x(n) as an impulse sequence, the following recursive equations are obtained.

$$y(n) = 2^{*}K1^{*}y(n-1) - y(n-2)$$
$$y(n-2) = y(n-1)$$
$$y(n-1) = y(n)$$

with the following conditions:

$$K1 = cos(wT)$$
$$K2 = initial\ y(n-1) = sin(wT)$$
$$K3 = initial\ y(n-2) = 0$$

IMPLEMENTATION

The above developed algorithm is implemented as a subroutine using a PIC17C42. All computations are performed using double precision arithmetic (16/32-bits). The recursive tone generation algorithm is implemented as a subroutine (labeled as `Resonator`). This subroutine generates samples of a single tone. To generate multiple frequencies, simply call this resonator routine for the desired frequencies and sum the outputs. The three tone co-efficients are stored in program memory and are read into the data memory using `TABLRD` instructions.

The fully commented code is listed in Appendix A. The timing and memory requirements are included in the comment sections of the code. For a listing of the header file `17C42.h` and the macro definition file `17C42.mac` please refer to Appendices C and D respectively of the application note DS00540. This code can be easily modified and used in various applications like DTMF generation, sound generation, etc. The tones generated can easily be output to an on-chip PWM channel which in turn can drive a speaker for producing various sounds. If using a PWM channel, set the PWM frequency much higher than the sampling frequency used (in the example code, for 8 kHz sampling frequency, use at least 20 kHz PWM frequency).

As an example, a dual tone is generated and the resulting digital wave form is analyzed. The main program calls the function `Resonator` twice for generating the two desired tones and the two outputs are summed. A sampling frequency of 8 kHz was used to generate a dual tone of 800 Hz and 1.10 kHz. The resulting wave form is shown in Figure 1. The spectrum of the signal shown in Figure 2 shows two peaks corresponding to the two desired tones (800 Hz and 1.10 kHz). The assembly code was tested using PICMASTER® (Microchip's Universal In-Circuit Emulator System).

© 1997 Microchip Technology Inc.

The generated tones were captured into the PICMASTER's trace buffer and then transferred to Microsoft® Excel® using Dynamic Data Exchange (DDE). Once the data is in Excel it is analyzed using Excel's FFT utility.

FIGURE 1: DUAL TONE WAVE FORM CAPTURED BY PICMASTER

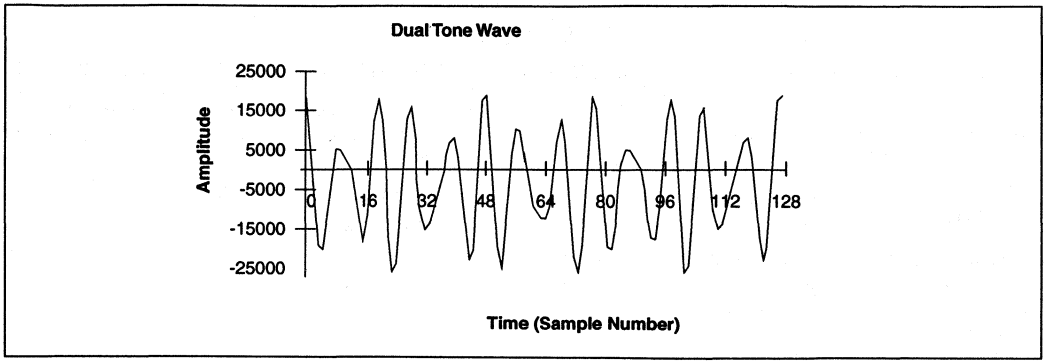
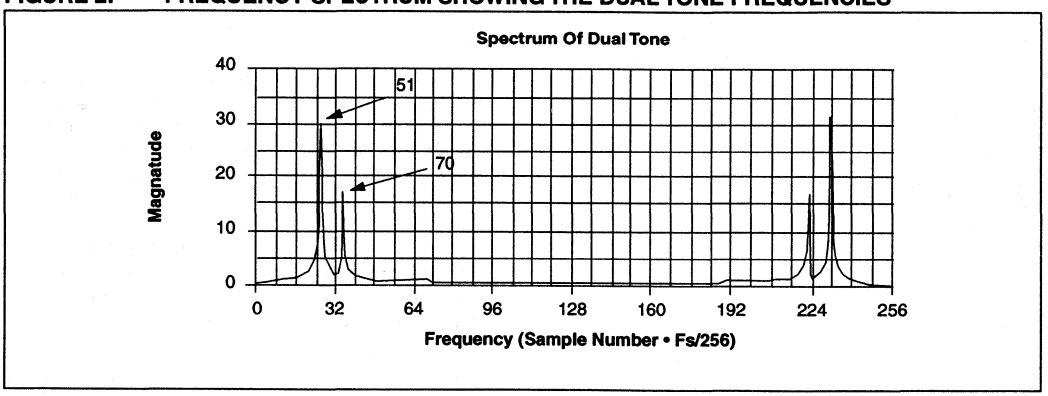


FIGURE 2: FREQUENCY SPECTRUM SHOWING THE DUAL TONE FREQUENCIES



PERFORMANCE

Table 1 below provides the performance of the resonator (labeled in the source code as `Resonator`) in terms of both timing and memory requirements. Since a double precision multiplier is used (software implementation), the multiplier timing is not always constant. Therefore the timings are given for the worst case. Note that irrespective of the frequency and the sampling (resolution) of the tone, program memory requirements is only 54 locations which in case of table lookup could be very large.

TABLE 0-1:

Cycles	235 Cycles
Time @ 16 MHz	58.75 µs
Time @ 25 MHz	47 µs
Data Memory	9 + 9* (# of tones to be generated)
Program memory	54 locations

APPLICATIONS

Tone generation is required in many applications. The code provided in this application note may be used as a general purpose routine to generate desired tones. It can be used in applications involving secure off-site control, where commands/data in the format of tones are transmitted over a telephone line. The tone generation finds applications involving signal modulations as well. The routine can be used to generate audible tones and output to a speaker connected to an I/O Port or a PWM channel.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code.
Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe®
(CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40 Released            TONE.ASM   1-16-1997  15:04:23         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P=17C42, columns=120,WRAP, R=DEC
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;                       Dual Tone Generation
                     00005 ;
                     00006 ;       A generic resonator subroutine is implemented to generate
                     00007 ; tones. Samples Of A Sin/Cos Wave are generated using
                     00008 ; recursive techniques. Table Lookups are thus avoided
                     00009 ; This is especially useful in generating multiple tones
                     00010 ; (e.g. DTMF tone generation, tone signalling, etc), or programmable
                     00011 ; tone generation which may vary for each application unit.
                     00012 ;
                     00013 ;
                     00014 ;       Program:          TONE.ASM
                     00015 ;       Revision Date:
                     00016 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00017 ;
                     00018 ;*****************************************************************
                     00019 ;
                     00020         include "p17c42.inc"
                     00001         LIST
                     00002 ;P17C42.INC Standard Header File, Ver. 1.03 Microchip Technology, Inc.
                     00264         LIST
                     00021
                     00022         #define TRUE    1
                     00023         #define FALSE   0
                     00024         #define LSB     0
                     00025         #define MSB     7
                     00026
                     00027         include "17c42.mac"
                     00945         LIST
                     00028 ;
                     00029 ;*****************************************************************
                     00030 ;       TBLADDR
                     00031 ;
                     00032 ; DESCRIPTION:
                     00033 ;       Load 16 bit table pointer with specified label
                     00034 ;
                     00035 ; TIMING (in cycles):
                     00036 ;       4
                     00037 ;
                     00038
                     00039 TBLADDR MACRO   label
                     00040
                     00041         MOVLW   (label)
                     00042         MOVWF   TBLPTRL
                     00043         MOVLW   page    (label)
                     00044         MOVWF   TBLPTRH
                     00045
                     00046         ENDM
                     00047
                     00048 ;*****************************************************************
                     00049 ;       ADDLBL
                     00050 ;
```


© 1997 Microchip Technology Inc.

```
                00051 ; DESCRIPTION:
                00052 ;       Add A Label (16 bit constant) To A File Register (16 bit)
                00053 ;
                00054 ; TIMING (in cycles):
                00055 ;       4
                00056 ;
                00057
                00058 ADDLBL  MACRO   label,f
                00059
                00060         MOVLW   (label)
                00061         ADDWF   f+BB0, F
                00062         MOVLW   page    (label)
                00063         ADDWFC  f+BB1, F
                00064
                00065         ENDM
                00066
                00067 ;*****************************************************************
                00068
                00069         LIST
                00070 ;
                00071         CBLOCK  0
00000000        00072                 BB0,BB1,BB2,BB3 ; RAM offset constants
                00073         ENDC
                00074 ;
                00075         CBLOCK  0x18
00000018        00076          AARG,AARG1                     ; 16 bit multiplier A
0000001A        00077          BARG,BARG1                     ; 16 bit multiplicand B
0000001C        00078          DPX,DPX1,DPX2,DPX3             ; 32 bit multiplier result = A*B
                00079          ENDC
                00080 ;
                00081         CBLOCK                         ; Generic Resonator Variables
00000020        00082          K1,K11                         ; K1
00000022        00083          SinCos_2C,SinCos_2,SinCos1_2   ; y(n-2), Init Value = K3
00000025        00084          SinCos_1,SinCos1_1             ; y(n-1), Init Value = K2
00000027        00085          SinCos,SinCos1                 ; y(n)
                00086
                00087         ENDC
                00088
                00089         CBLOCK
00000029        00090          tone1K1,tone1K11               ; Tone 1 variables
0000002B        00091          tone1_2C,tone1_2,tone11_2
0000002E        00092          tone1_1,tone11_1
00000030        00093                 tone1,tone11
                00094
00000032        00095          tone2K1,tone2K11              ; Tone 2 variables
00000034        00096          tone2_2C,tone2_2,tone21_2
00000037        00097          tone2_1,tone21_1
00000039        00098          tone2,tone21
                00099
0000003B        00100           dualTone,dualTone1           ; tone1+tone2
                00101         ENDC
                00102
                00103 ;
                00104 ;*********************************************************
                00105 ;
                00106 #define ADD_OFFSET  FALSE
                00107 ;
                00108 #define coeff_addr  0x0030  ; prog mem addr of sine wave frequency,etc
                00109 #define DummyAddr   0x0A00
                00110 ;
                00111 ;*****************************************************************
                00112 ;               Test Routine For Sin Wave Generation
                00113 ;
                00114 ;       (a) Call Initialization Subroutine to read desired sine wave
                00115 ;               freq from prog mem to data mem.
                00116 ; (b) Call the resonator subroutine to generate samples of desired
```


© 1997 Microchip Technology Inc.

```
                    00117 ; sine wave.
                    00118 ; (c) Perform a Dummy Table Write of The Sine Wave Data, so that
                    00119 ; PIC-MASTER emulator can capture the data into trace buffer
                    00120 ;
                    00121 ;*****************************************************************
                    00122 ;
0000                00123         ORG     0x0000
0000 C040           00124         goto    start
                    00125 ;
                    00126 ;*****************************************************************
                    00127 ;               Resonator Co-efficients For Desired Tones
                    00128
0030                00129    ORG     coeff_addr
                    00130
                    00131 ; Tone 1 Resonator Constants
                    00132 ; Sample Rate = fs = 8 khz, Tone Freq = f = 0.800 Khz
                    00133 ;
0030 678E           00134   DATA    26510  ; K1 = COS(wT) = COS(360*f/fs)
0031 259E           00135   DATA    9630   ; K2 = SIN(wT) = SIN(360*f/fs) = init value of y(n-1)
0032 0000           00136   DATA    0      : K3 is init value of y(n-2)
                    00137 ;
                    00138 ; Tone 2 Resonator Constants
                    00139 ; Sample Rate = fs = 8 khz, Tone Freq = f = 1.10 Khz
                    00140 ;
0033 5321           00141   DATA    21281  ; K1 = COS(wT) = COS(360*f/fs)
0034 1855           00142   DATA   6229    ; K2 = SIN(wT) = SIN(360*f/fs) = init value of y(n-1)
0035 0000           00143   DATA    0      ; K3 is init value of y(n-2)
                    00144 ;
                    00145 ;*****************************************************************
                    00146 ;
0040                00147   ORG     0x0040
0040                00148 start
0040 E057           00149    call    Coeff_Read
                    00150 ; load table pointers with a dummy  addr for PIC-MASTER data capture
                    00151         MOVK16  DummyAddr,TBLPTRL
                        M
0041 B000               M   MOVLW   (0x0A00) & 0xff
0042 010D               M   MOVWF   TBLPTRL+B0
0043 B00A               M   MOVLW   ((0x0A00) >> 8)
0044 010E               M   MOVWF   TBLPTRL+B1
                        M
0045                00152 NextSample
0045 0000           00153   nop
0046 A43B           00154   tlwt    0,dualTone
0047                00155 capture
0047 AE3C           00156   tablwt  1,0,dualTone+BB1   ; for PIC-MASTER trace capture
0048 0000           00157   nop
                    00158 ;
0049 B029           00159   movlw   tone1K1            ; load indirect address pointer
004A 0101           00160   movwf   FSR0               ; for Tone 1
004B E06C           00161   call    Resonator          ; Compute next sample of tone 1
                    00162
004C B032           00163   movlw   tone2K1            ; load indirect address pointer
004D 0101           00164   movwf   FSR0               ; for Tone 2
004E E06C           00165   call    Resonator          ; compute next sample of tone 2
                    00166 ;
                    00167 ; Compute Tone1 + Tone2 for dual tone generation
                    00168 ;
                    00169   ADD16ACC tone1,tone2,dualTone  ; dualTone = tone1 + tone2
                        M
004F 6A30               M   movfp   tone1+B0,WREG
0050 0E39               M   addwf   tone2+B0,W
0051 013B               M   movwf   dualTone+B0
0052 6A31               M   movfp   tone1+B1,WREG
0053 103A               M   addwfc  tone2+B1,W
0054 013C               M   movwf   dualTone+B1
```

```
                        M
                    00170 ;
0055 C045           00171    goto    NextSample
                    00172 ;
0056 C056           00173 self  goto self
                    00174 ;
                    00175 ;*****************************************************************
                    00176 ; Initialization routine :
                    00177 ; Read Tone 1 & Tone 2 Resonator Frequencies from Program Memory To
                    00178 ; Data Memory
                    00179 ;
                    00180 ;       Program Memory  :       3 + 8*(#of tones to be generated)
                    00181 ;       Timing          :       4 + 11*(#of tones to be generated)
                    00182 ;*****************************************************************
                    00183
0057                00184 Coeff_Read
                    00185
                    00186    TBLADDR coeff_addr
                        M
0057 B030               M    MOVLW   (0x0030)
0058 010D               M    MOVWF   TBLPTRL
0059 B000               M    MOVLW   page    (0x0030)
005A 010E               M    MOVWF   TBLPTRH
                        M
005B A929           00187    tablrd  0,1,tone1K1
005C A029           00188    tlrd    0,tone1K1
005D AB2A           00189    tablrd  1,1,tone1K1+BB1 ; read K1
005E A02E           00190    tlrd    0,tone1_1
005F AB2F           00191    tablrd  1,1,tone1_1+BB1 ; read K2
0060 A02C           00192    tlrd    0,tone1_2
0061 A22D           00193    tlrd    1,tone1_2+BB1   ; read K3
0062 292B           00194    clrf    tone1_2C, F
                    00195
0063 A932           00196    tablrd  0,1,tone2K1
0064 A032           00197    tlrd    0,tone2K1
0065 AB33           00198    tablrd  1,1,tone2K1+BB1 ; read K1
0066 A037           00199    tlrd    0,tone2_1
0067 AB38           00200    tablrd  1,1,tone2_1+BB1 ; read K2
0068 A035           00201    tlrd    0,tone2_2
0069 A236           00202    tlrd    1,tone2_2+BB1   ; read K3
006A 2934           00203    clrf    tone2_2C, F
                    00204
006B 0002           00205    return
                    00206 ;
                    00207 ;*******************************************************
                    00208 ;                       Resonator Subroutine
                    00209 ;
                    00210 ; Before calling this routine, load the indirect register, FSR0
                    00211 ; with the starting RAM address of the desired Tone Variables,
                    00212 ; ( eg. For Tone1 Generation, load FSR0 with "Tone1K1" address.
                    00213 ;
                    00214 ;
                    00215 ;
                    00216 ; Timing (worst case) :
                    00217 ;               20 + 36 + (worst case multiplier time)
                    00218 ;               = 56 + 179 = 235 cycles
                    00219 ;               = 58.75 uS @ 16Mhz
                    00220 ;               = 47.00 uS @ 20Mhz
                    00221 ;               = 37.60 uS @ 25 Mhz
                    00222 ;
                    00223 ; Memory Requirements :
                    00224 ;               Program Memory  : 54 locations
                    00225 ;               Data Memory     : 9 + 9*(#of tones to be generated)
                    00226 ;
                    00227 ;*******************************************************
                    00228 ;
```


© 1997 Microchip Technology Inc.

```
                    00229
006C                00230 Resonator
                    00231
                    00232 ;
                    00233 ; Transfer tone variables to resonator's variables using indirect
                    00234 ; addressing This is necessary for making the "Resonator" a generic
                    00235 ; subroutine, so that the same code can be called for various
                    00236 ; tones.
                    00237 ; Indirect addressing mode can be used throught the code in this
                    00238 ; subroutine, but is less efficient.
                    00239 ;
006C 8D04           00240   bcf     ALUSTA,FS1
006D 8404           00241   bsf     ALUSTA,FS0              ; auto increment FSR0
006E 4020           00242   movpf   INDF0,K1+BB0
006F 4021           00243   movpf   INDF0,K1+BB1
0070 4022           00244   movpf   INDF0,SinCos_2C
0071 4023           00245   movpf   INDF0,SinCos_2+BB0
0072 4024           00246   movpf   INDF0,SinCos_2+BB1
0073 4025           00247   movpf   INDF0,SinCos_1+BB0
0074 4026           00248   movpf   INDF0,SinCos_1+BB1
0075 4027           00249   movpf   INDF0,SinCos+BB0
0076 4028           00250   movpf   INDF0,SinCos+BB1
                    00251
                    00252 ;
                    00253 ; Compute 2*K1*y(n-1)
                    00254 ;
                    00255   MOVFP16 K1,BARG
                        M
0077 7A20               M   MOVFP   K1+B0,BARG+B0           ; move A(B0) to B(B0)
0078 7B21               M   MOVFP   K1+B1,BARG+B1           ; move A(B1) to B(B1)
                        M
                    00256   MOVFP16 SinCos_1,AARG
                        M
0079 7825               M   MOVFP   SinCos_1+B0,AARG+B0     ; move A(B0) to B(B0)
007A 7926               M   MOVFP   SinCos_1+B1,AARG+B1     ; move A(B1) to B(B1)
                        M
007B E0A2           00257   call    DblMult
                    00258   RLC32   DPX                     ; DPX = 2*K1*y(n-1)
                        M
007C 8804               M   BCF     ALUSTA,C
007D 1B1C               M   RLCF    DPX+B0, F
007E 1B1D               M   RLCF    DPX+B1, F
007F 1B1E               M   RLCF    DPX+B2, F
0080 1B1F               M   RLCF    DPX+B3, F
                        M
                    00259 ;
                    00260 ; subtract y(n-2)*(2**15)
                    00261 ;
0081 2922           00262   clrf    SinCos_2C, F
                    00263   RRC24   SinCos_2C
                        M
0082 1A24               M   RLCF    SinCos_2C+B2,W          ; move sign into carry bit
0083 1924               M   RRCF    SinCos_2C+B2, F
0084 1923               M   RRCF    SinCos_2C+B1, F
0085 1922               M   RRCF    SinCos_2C+B0, F
                        M
                    00264   SUB24   SinCos_2C,DPX1          ; DPX = 2*K1*y(n-1) - y(n-2)
                        M
0086 6A22               M   MOVFP   SinCos_2C+B0,WREG       ; get lowest byte of a into w
0087 051D               M   SUBWF   DPX1+B0, F              ; sub lowest byte of b, save in b(B0)
0088 6A23               M   MOVFP   SinCos_2C+B1,WREG       ; get 2nd byte of a into w
0089 031E               M   SUBWFB  DPX1+B1, F              ; sub 2nd byte of b, save in b(B1)
008A 6A24               M   MOVFP   SinCos_2C+B2,WREG       ; get 3rd byte of a into w
008B 031F               M   SUBWFB  DPX1+B2, F              ; sub 3rd byte of b, save in b(B2)
                        M
                    00265
```


© 1997 Microchip Technology Inc.

```
                    00266   RLC24   DPX1                    ; adjust decimal point
                        M
008C 8804               M   BCF     ALUSTA,C
008D 1B1D               M   RLCF    DPX1+B0, F
008E 1B1E               M   RLCF    DPX1+B1, F
008F 1B1F               M   RLCF    DPX1+B2, F
                        M
                    00267 ;
                    00268 ; update past samples with newly computed values
                    00269 ;
                    00270   MOVPF16 DPX2,SinCos             ; y(n) = 2*K1*y(n-1) - y(n-2)
                        M
0090 5E27               M   MOVPF   DPX2+B0,SinCos+B0       ; move A(B0) to B(B0)
0091 5F28               M   MOVPF   DPX2+B1,SinCos+B1       ; move A(B1) to B(B1)
                        M
                    00271   MOV16   SinCos_1,SinCos_2       ; y(n-2) = y(n-1)
                        M
0092 6A25               M   MOVFP   SinCos_1+B0,WREG        ; get byte of a into w
0093 0123               M   MOVWF   SinCos_2+B0             ; move to b(B0)
0094 6A26               M   MOVFP   SinCos_1+B1,WREG        ; get byte of a into w
0095 0124               M   MOVWF   SinCos_2+B1             ; move to b(B1)
                        M
                    00272   MOVPF16 DPX2,SinCos_1           ; y(n-1) = y(n)
                        M
0096 5E25               M   MOVPF   DPX2+B0,SinCos_1+B0     ; move A(B0) to B(B0)
0097 5F26               M   MOVPF   DPX2+B1,SinCos_1+B1     ; move A(B1) to B(B1)
                        M
                    00273 ;
                    00274 ; Generation Of The Next Sample Of The Resonator (sine wave) is done.
                    00275 ; The 16 bit result is stored in location "SinCos" (low Byte) &
                    00276 ; "SinCos+1" (High Byte)
                    00277 ;
                    00278 ; write back all the computed values to respective tone variables
                    00279 ;
0098 0701           00280   decf    FSR0, F
0099 8D04           00281   bcf     ALUSTA,FS1
009A 8C04           00282   bcf     ALUSTA,FS0
009B 6028           00283   movfp   SinCos+BB1,INDF0
009C 6027           00284   movfp   SinCos+BB0,INDF0
009D 6026           00285   movfp   SinCos_1+BB1,INDF0
009E 6025           00286   movfp   SinCos_1+BB0,INDF0
009F 6024           00287   movfp   SinCos_2C+BB2,INDF0
00A0 6023           00288   movfp   SinCos_2C+BB1,INDF0
                    00289 ;
00A1 0002           00290   return
                    00291 ;
                    00292 ;*****************************************************************
                    00293 ;       Include Double Precision Multiplication Routine
                    00294 ;*****************************************************************
  00000001          00295 SIGNED  equ     TRUE
                    00296
                    00297    include "17c42mpy.mac"
                    00178    LIST
                    00298
                    00299 ;
                    00300    END
```


© 1997 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X--------------- ---------------- ---------------- XXXXXX----------
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXX------------

All other memory blocks unused.

Program Memory Words Used:   379


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN544

Math Utility Routines

Author: *Amar Palacherla*
Microchip Technology Inc.

INTRODUCTION

PLEASE NOTE: *This application note uses the old Microchip Math Routine format. It is intended for reference purposes only and is being provided for those of you still implementing Binary Coded Decimal(BCD) routines. For any new designs, please refer to application notes contained in Microchip's Embedded Control Handbook Volume II - Math Library*

This application note provides some utility math routines for Microchip's second generation of high performance 8-bit microcontroller, the PIC17C42. Three assembly language modules are provided, namely ARITH.ASM, BCD.ASM and FXP–DIV.ASM. Currently in each file the following subroutines are implemented:

ARITH.ASM

- Single precision 8 x 8 unsigned multiply
- 16 x 16 double precision multiply (signed or unsigned)
- 16 / 16 double precision divide (signed or unsigned)
- 16 x 16 double precision addition
- 16 x 16 double precision subtraction
- double precision square root
- double precision numerical differentiation
- double precision numerical integration
- Pseudo Random number generation
- Gaussian distributed random number generation

BCD.ASM

- 8-bit binary to 2 digit BCD conversion
- 16-bit binary to 5 digit BCD conversion
- 5-bit BCD to 16-bit binary conversion
- 2 digit BCD addition

FXP–DIV.ASM.

The routines that are implementing in this source file are shown in Table 3.

As more routines are available, they will be added to the library. The latest routines may be obtained either through Microchip's bulletin board or by contacting your nearest Microchip sales office for a copy on a MS-DOS® floppy.

These routines have been optimized wherever possible with a compromise between speed, RAM utilization, and code size. Some routines (multiplication and division) are provided in two forms, one optimized for speed and the other optimized for code size.

All the routines have been implemented as callable subroutines and the usage of each routine is explained below. At the end of the application note, the listing files of the above programs are given.

4

SINGLE PRECISION UNSIGNED MULTIPLICATION (8 x 8)

This routine computes the product of two unsigned 8-bit numbers and produces a 16-bit result. Two routines are provided: one routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 1.

DOUBLE PRECISION MULTIPLICATION

This routine computes the product of 16- bit numbers and produces a 32-bit result. Both signed and unsigned arithmetic is provided (2's complement arithmetic). Whether to use signed or unsigned is decided at assembly time depending on whether "SIGNED" is set to true or false (refer to the source code). These routines are extremely useful for high precision computation and are used extensively in the other programs provided in this application note (for example, the square root, integrator, differentiator call these routines). Two routines are provided. One routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 2.

TABLE 1: SINGLE PRECISION MULTIPLICATION

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
mpy8x8_F	speed efficient	36	36	0	used
MPY8X8_s	code efficient	13	69	1	used

MS-DOS is a registered trademark of Microsoft Corporation.

© 1997 Microchip Technology Inc.

The listing file shown is assembled with "SIGNED equ TRUE". If unsigned arithmetic is needed, the source code should be changed to "SIGNED equ FALSE". Conditional assembly and the advanced macro features of the assembler are used.

The data memory organization is explained in the comment section of the code. Faster execution and code space saving can be achieved by setting "MODE_FAST equ TRUE". However, setting MODE_FAST variable to TRUE restricts that operands and the 32-bit result be in data RAM locations 0x18 and 0x1F (in this mode, MOVFP and MOVPF instructions may be used to transfer data to/from any RAM location to addresses less than 0x1F). If MODE_FAST is set to FALSE, there will be no restriction on the location of the data RAM values used in this subroutine. However, the code will be slightly slower and occupies more program memory.

The listing file shown is assembled with "SIGNED equ TRUE". If unsigned arithmetic is needed, the source code should be changed to "SIGNED equ FALSE". Conditional assembly and the advanced macro features of the assembler are used.

DOUBLE PRECISION DIVISION

This routine performs a 2's complement division of two 16-bit numbers and produces a 16-bit quotient with a 16-bit remainder. Both signed and unsigned arithmetic is provided (2's complement arithmetic). Whether to use signed or unsigned is decided at assembly time depending on whether "SIGNED" is set to true or false (refer to the source code).

These routines are extremely useful for high precision computation and are used extensively in the other programs provided in this application note (for example, the square root, integrator, differentiator call these routines). Two routines are provided. One routine is optimized for speed (a straight line code) and the other one has been optimized for code size (a looped code version). These subroutines are located in Appendix C. The performance specs are shown in Table 3.

TABLE 2: DOUBLE PRECISION MULTIPLICATION

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
D_myfF	Speed Efficient, Signed Arithmetic	204	183	1	used
D_mpyF	Speed Efficient, Unsigned Arithmetic	179	176	0	used
D_mpyS	Code Efficient, Signed Arithmetic	52	254	4	used
D_mpyS	Code Efficient, Unsigned Arithmetic	21	242	3	used

TABLE 3: FIXED POINT DIVIDE PERFORMANCE DATA

Routine	Max. Cycles	Min. Cycles	Program Memory	Data Memory
16 / 8 Signed	146	135	146	5
16 / 8 Unsigned	196	156	195	4
16 / 7 Unsigned	130	130	129	4
15 / 7 Unsigned	125	125	124	4
16 / 16 Unsigned	214	187	241	7
16 / 16 Unsigned	244	180	243	6
16 / 15 Unsigned	197	182	216	6
16 / 15 Unsigned	191	177	218	6
32 / 16 Unsigned	414	363	476	9
32 / 16 Unsigned	485	459	608	9
32 / 15 Unsigned	390	359	451	8
31 / 15 Unsigned	383	353	442	8

© 1997 Microchip Technology Inc.

DOUBLE PRECISION ADDITION AND SUBTRACTION

Two routines are provided. One performs a 2's complement addition and the other one performs a 2's complement subtraction of two 16-bit binary numbers. These subroutines are located in ARITH.ASM and printed in the listing file ARITH.LST. The performance specs are shown in Table 4.

NEGATE A DOUBLE PRECISION NUMBER

These routines negate a double precision number (16-bit and 32-bit). Two routines and two macros are provided to negate a 16-bit number. The subroutines use indirect addressing mode and the macros use a direct addressing scheme. A macro is provided to negate a 32- bit number.

DOUBLE PRECISION SQUARE ROOT

Often in many applications, one needs to find the square root of a number. Of the many numerical methods available to compute the square root of a number, the Newton-Raphson method is one of the most attractive because of its fast convergence rate. In this method, the square root of number, N, is obtained as an approximate solution of

$$f(Y) = Y^2 - N = 0$$

The function f(Y) can be expanded about Y_0 using the first order Taylor polynomial expansion as:

EQUATION 1:

$$f(Y) = f(Y_0) + (Y - Y_0)f'(Y_0) + \frac{(Y - Y_0)2f''f(Y_0)}{2!} + \ldots$$

If X is a root of f(Y), then f(X) = 0: Therefore,

If Y0 is an approximate root of f(Y), then the higher order terms in the above equation are negligible.

Therefore, $f(Y_0) + (X - Y_0)f(Y_0) = 0$

i.e., $X = Y_0 + \frac{f(Y_0)}{f'(Y_0)}$

Thus X is a better approximation for Y_0. From the previous equation, the sequence {Xn} can be generated:

EQUATION 2: $$X_n = X_{n-1} - \frac{f(X_{n-1})}{f'(X_{n-1})}, n \geq 1$$

For our case, equation 2, reduces to:

EQUATION 3: $$\frac{X_{n-1} + \frac{N}{X_{n-1}}}{2}$$

The routine "Sqrt" in ARITH.ASM implements the above equation. Equation 3 requires that at first an initial approximation for the root is known. The better the initial approximation, the faster the convergence rate would be. In the "Sqrt" routine, the initial approximation root is set as N/2. This routine calls the double precision division routine (D_divS).

In the code size, the Division routine (Ddiv_S) size is not included.

TABLE 4: DOUBLE PRECISION ADDITION AND SUBTRACTION

Name	Program Memory	Instruction Cycles	Scratch RAM	W Register
Dadd	4	4	0	used
Dsub	4	4	0	used

TABLE 5: NEGATE A DOUBLE ADDITION AND SUBTRACTION

Name	Program Memory	Instruction Cycles	Scratch RAM	W Register
Negate	7	7	0	unused
NegateAlt	7	7	0	used
NegMac	5	5	0	used
AltNegMac	5	5	0	unused
NegMac32 (32 bit)	11	11	0	used

© 1997 Microchip Technology Inc.

BCD ROUTINES

Three routines are provided for general purpose BCD arithmetic:

a) BCD to binary conversion
b) Binary to BCD conversion
c) BCD addition

The BCD to binary conversion routine converts a 5-digit BCD code to a 16-bit binary number. The BCD addition routine adds two BCD digits directly without converting them at first to binary. Note the usage of the **"DAW"** instruction. The other two routines convert a binary number to a BCD code. The performance specs for the BCD routines is given in the Table 7 below.

NUMERICAL DIFFERENTIATION

This routine performs numerical differentiation of a sequence of data if the input sequence is assumed to be piecewise linear with no discontinuances (this is the case in most real world signals). Although this routine is provided as a tool to implement a PID algorithm for motor control, it can be used as a general purpose subroutine. This routine uses the so called 3-Point formula to compute the differential of a sequence of numbers.

Given an equation f(t), its derivative is given by

$$f'(t) = \frac{df(t)}{dt}$$

The above equation can be approximated using the 3-Point formula as given below:

EQUATION 4: 3-POINT FORMULA:

$$f'(t) = \frac{df(t)}{dt} = \frac{1}{2h}[f(t_0 - 2h) - 4f(T_0 - h) + 3f(t_0)]$$

where t_0 is the point at which the numerical derivative is desired and "h" is the step size. The smaller the value of the step size (h), the better the approximation. In case of say, PID motor control, the step size is proportional to the time intervals at which the new sample value of the position (or speed) is obtained. Using the above equation to compute the differential, three samples are necessary (present value and the last two past values). The subroutine "Diff" is implemented so that 1/2h factor is stored already in a RAM location (location DiffK) as 1/2h and not as "h" because it is more efficient to multiply than divide.

After computation, the routine does not move the present value to the past value. So the user must update the past values before calling this routine again. This way, if necessary, differentiation may be performed without disturbing the present and past values. Also, when this routine is called for the first time, it is user's responsibility to set the initial values of the past data points (may be set to zero). This routine called "Diff" is located in "ARITH.ASM".

In the code size, the double precision multiplication routine (Dmpy_S) used is not included.

TABLE 6: DOUBLE PRECISION SQUARE ROOT

Name	Program Memory	Instruction Cycles	Scratch RAM	W Register
Sqrt	22	3300 (Approx.)	6	used

TABLE 7: BCD ROUTINES

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
BCDtoB	BCD to Binary	30	112	0	used
B2_BCD_Looped	Binary to BCD (16 bit) looped code	32	750	1	used
B2_BCD_Straight	Binary to BCD (16 bit) straight line code	44	572	1	used
BinBCD	Binary to BCD (8 bit)	10	62	1	unused
BCDAdd	BCD addition	5	5	0	used

© 1997 Microchip Technology Inc.

NUMERICAL INTEGRATION

This routine performs numerical integration using Simpson's Three-Eighths Rule. This is a third order approximation for the function, whose integral is to be computed at a given point. Although this routine is provided as a tool to implement a PID algorithm for motor control, it can be used as a general purpose subroutine. Given a function f(t), its integral over a range t_0 to t_3 is represented as:

t3

$\int$f(t)dt. This function is approximated as follows:

t0

Simpson's Three-Eighths Rule:

t3

$$\int f(t)dt = \frac{3h}{8}[f(t_0) + 3f(t1) + 3f(t2) + f(t3)]$$

(t_0)

The constant 3h/8 can be computed before hand and stored in a RAM location (in location IntgKLo and IntgKHi as a 16-bit number). After computation, the routine does not move the present value to the past value. So the user must update the past values before calling this routine again. This way, if necessary, integration may be performed without disturbing the present and past values. Also, when this routine is called for the first time, it is user's responsibility to set the initial values of the past data points (may be set to zero). This routine called "Integrate" is located in "ARITH.ASM".

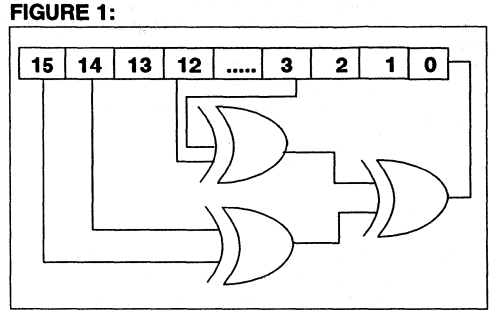
In the code size, the double precision multiplication routine (Dmpy_S) used is not included.

PSEUDO RANDOM NUMBER GENERATOR

This routine (subroutine "Random 16" provided in ARITH.ASM) generates a pseudo random number sequence. The random points are generated using a 16-bit register and left shifting the contents with the LSB set as shown by the following schematic.

As a test, the random points are generated by calling the subroutine from an infinite loop, and the data points are continuously captured into the real time trace buffer using the PICMASTER (the Universal In-Circuit Emulator for the PICmicro™ series). The autocorrelation of the captured data is computed using a stand alone program and is shown in Figure 2. From this figure, it can be seen that the data has a strong autocorrelation only at the origin and sharply approaches to zero within a few points. This demonstrates the randomness of the data captured.

FIGURE 1:



4

TABLE 8: DIFFERENTIATION

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
Diff	Numerical Differentiation	34	365	10	used

TABLE 9: INTEGRATION

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
Integrate	Numerical Integration	39	370	12	used

© 1997 Microchip Technology Inc.

FIGURE 2: AUTOCORRELATION OF THE DATA POINTS GENERATED BY THE RANDOM NUMBER GENERATOR

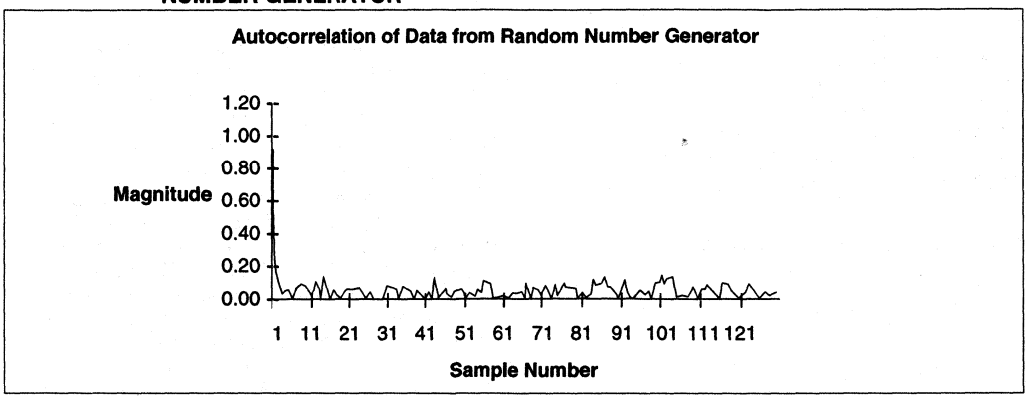
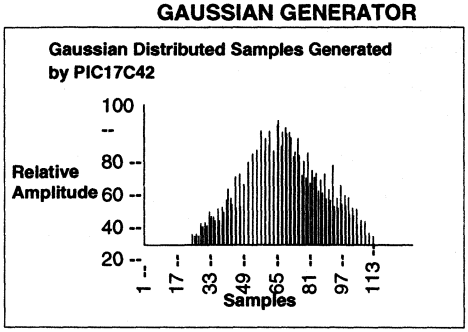


TABLE 10: RANDOM DOUBLE GENERATOR

Name	Comments	Program Memory	Instruction Cycles	Scratch RAM	W Register
Random16	Pseudo Random Number Generator	12	12	0	used
Gauss	Gaussian Random Number Generator	21	452	4	used

PN (pseudo noise) sequences are widely used in digital communication systems for synchronization. These code words can also be used for data scrambling because of their good correlation properties. An interesting application of these sequences is system integrity. For example, these sequences can be regularly transmitted to a processor whose watchdog timer will time out if, say, two consecutive PN sequences do not match.

FIGURE 3: HISTOGRAM OF THE DATA GENERATED BY THE GAUSSIAN GENERATOR



GAUSSIAN DISTRIBUTED RANDOM NUMBER GENERATOR

This routine (subroutine "Gauss" provided in ARITH.ASM) generates a sequence of random numbers with a characteristic of a normal distribution (Gaussian distributed points). This routine calls the pseudo random number generator ("random16") to obtain a near uniformly distributed random points and from these points, the Gaussian distributed points are generated. The method of generating Gaussian points is based on the "Central Limit Theorem", which states that an ensemble of average weighted sum of a sequence of uncorrelated samples tends to have a Gaussian distribution.

As a test, the Gaussian points are generated by calling the subroutine from an infinite loop, and the data points are continuously captured into the real time trace buffer using the PICMASTER (the Universal In-Circuit Emulator for the PICmicro series). A plot of the points captured is shown in Figure 3, which shows that the random points generated have the characteristics of a Gaussian distribution.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: GENERAL PURPOSE MATH ROUTINES LISTING FILE OF ARITH.LST

```
MPASM 01.40 Released            ARITH.ASM   1-16-1997  15:10:04         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001
                     00002         TITLE   "General Purpose Math Routines For PIC17C42 : Ver 1.0"
                     00003
                     00004         LIST    P = 17C42, columns=120, WRAP, L=0, R = DEC
                     00005 ;
                     00006         include      <p17c42.inc>
                     00001         LIST
                     00002 ; P17C42.INC  Standard Header File, Version 1.03 Microchip Technology, Inc.
                     00264         LIST
                     00007
                     00008         #define TRUE    1
                     00009         #define FALSE   0
                     00010
00000001             00011 _INC     equ     1
00000000             00012 _NO_INC equ      0
00000000             00013 _LOW     equ     0
00000001             00014 _HIGH    equ     1
                     00015
                     00016 ;
                     00017 ;*****************************************************************
                     00018 ;       Define RAM Locations necessary For the "ARITH.ASM"
                     00019 ;   RAM locations should be defined before calling the library math
                     00020 ; routines
                     00021 ;
                     00022 ;       Program:          ARITH.ASM
                     00023 ;       Revision Date:
                     00024 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00025 ;
                     00026 ;*****************************************************************
                     00027 ;
00000001             00028 MODE_FAST        equ     TRUE
00000000             00029 SIGNED           equ     FALSE
                     00030 ;
                     00031 ;*****************************************************************
                     00032 ;
                     00033  #if MODE_FAST
                     00034
                     00035   CBLOCK  0x18
00000018             00036      ACCaLO, ACCaHI, ACCbLO, ACCbHI      ; Ram Locations for Arithmetic
0000001C             00037      ACCcLO, ACCcHI, ACCdLO, ACCdHI      ;      Routines
                     00038   ENDC
                     00039
                     00040  #else
                     00041
                     00042   CBLOCK  0x20
                     00043      ACCaLO, ACCaHI, ACCbLO, ACCbHI
                     00044      ACCcLO, ACCcHI, ACCdLO, ACCdHI
                     00045   ENDC
                     00046
                     00047  #endif
                     00048 ;
                     00049
                     00050   CBLOCK
```

```
00000020  00051       tempLo, tempHi, count, sign
          00052     ENDC
          00053
          00054     CBLOCK
00000024  00055           NumLo, NumHi
00000026  00056           iterCnt
          00057     ENDC
          00058 ;
          00059           CBLOCK                          ; RAM locations for "Diff" routine
00000027  00060                   XnLo, XnHi, Xn_1_Lo
0000002A  00061                   Xn_1_Hi, Xn_2_Lo, Xn_2_Hi
0000002D  00062                   DiffKLo, DiffKHi                ; DiffK = h = Step Size
0000002F  00063                   DiffLo, DiffHi
          00064           ENDC
          00065 ;
          00066           CBLOCK                          ; RAM Locations for "Integrate"
00000031  00067                   X0Lo, X0Hi, X1Lo, X1Hi  ; Routine
00000035  00068                   X2Lo, X2Hi, X3Lo, X3Hi
00000039  00069                   IntgKLo, IntgKHi          ; INTEGRATE CONST = 3*h/8
0000003B  00070                   IntgLo, IntgHi
          00071           ENDC
          00072 ;
          00073 ;*****************************************************************
          00074 ;
00000018  00075 mulcnd  equ     ACCaLO
00000019  00076 mulplr  equ     ACCaHI
0000001A  00077 L_byte  equ     ACCbLO
0000001B  00078 H_byte  equ     ACCbHI
          00079 ;
0000000A  00080 _LUPCNT  equ    10              ; Set Desired Number of iterations
0000001E  00081 SqrtLo  equ     ACCdLO      ; for Square Root Routine(NEWTON Iterations)
0000001F  00082 SqrtHi  equ     ACCdHI
          00083 ;
          00084 ;  Define RAM locations for the Random Number Generators
          00085 ;
00000018  00086 RandLo  equ     ACCaLO
00000019  00087 RandHi  equ     ACCaHI    ; 16 bit Pseudo Random Number
0000001B  00088 GaussHi equ     ACCbHI
0000001A  00089 GaussLo equ     ACCbLO ; 16 bit Gaussian distributed number
00000020  00090 GaussTmp equ    tempLo
          00091 ;
          00092           PAGE
          00093
0000      00094           ORG     0x0000
          00095 ;*****************************************************************
          00096 ;                       Math Routines Test Program
          00097 ;*****************************************************************
          00098 ;
          00099 ;    Load constant values to ACCa & ACCb for testing
          00100 ;
0000      00101 main
0000 E02D 00102           call    loadAB          ; result of adding ACCb+ACCa->ACCb
0001 E036 00103           call    D_add           ; Here Accb = 81FE
          00104 ;
0002 E02D 00105           call    loadAB          ; result of subtracting ACCb - ACCa->ACCb
0003 E03B 00106           call    D_sub           ; Here Accb = 7E00
          00107 ;
0004 E02D 00108           call    loadAB          ; result of multiplying ACCb*ACCa->(ACCd,ACCc)
0005 E050 00109           call    D_mpyS          ; Here (ACCd,ACCc) = 00FF 7E01
          00110 ;
0006 E02D 00111           call    loadAB          ; result of multiplying ACCb*ACCa->(ACCd,ACCc)
0007 E065 00112           call    D_mpyF          ; Here (ACCd,ACCc) = 00FF 7E01
          00113 ;
0008 E02D 00114           call    loadAB          ; result of multiplying ACCb/ACCa->(ACCd,ACCc)
0009 E119 00115           call    D_divS          ; Here (ACCd,ACCc) = 0040 003f
          00116 ;
```


© 1997 Microchip Technology Inc.

```
000A E02D     00117         call    loadAB          ; result of multiplying ACCb/ACCa->(ACCd,ACCc)
000B E138     00118          call    D_divF          ; Here (ACCd,ACCc) = 0040 003f
              00119 ;
000C B0F3     00120         movlw   xf3
000D 0125     00121         movwf   NumHi
000E B0F6     00122         movlw   xf6             ; Set input test number = 62454
000F 0124     00123         movwf   NumLo           ;  = F3F6h
0010 E27D     00124         call    Sqrt            ; result = 00F9h = 249 (in SqrtLo)
              00125 ;                               ; exact sqrt(62454) = 249.9
              00126 ;
0011 B0FF     00127         movlw   0xff
0012 0119     00128         movwf   mulplr          ; multiplier (in mulplr) = 0FF
0013 B0FF     00129         movlw   0xff            ; multiplicand(W Reg )   = 0FF
0014 0118     00130         movwf   mulcnd
0015 E293     00131         call    mpy8x8_F        ; The result 0FF*0FF = FE01 is in locations
              00132 ;                               ; H_byte & L_byte
0016 B0FF     00133         movlw   0xff
0017 0119     00134         movwf   mulplr          ; multiplier (in mulplr) = 0FF
0018 B0FF     00135         movlw   0xff            ; multiplicand(W Reg )   = 0FF
0019 0118     00136         movwf   mulcnd
001A E2B8     00137         call    mpy8x8_S        ; The result 0FF*0FF = FE01 is in locations
              00138 ;                               ; H_byte & L_byte
              00139 ; Test The Random Number Generators
              00140 ; Capture data into trace buffer by TABLE WRITES to a
              00141 ; dummy Program Memory location
              00142 ;
001B B0FF     00143         movlw   0xff
001C 010D     00144         movwf   TBLPTRL
001D B05F     00145         movlw   0x5f
001E 010E     00146         movwf   TBLPTRH
              00147 ;
001F B030     00148         movlw   0x30
0020 0119     00149         movwf   RandHi
0021 B045     00150         movlw   0x45
0022 0118     00151         movwf   RandLo
              00152 ;
0023 C028     00153         goto    GaussPoint
              00154 ;
0024          00155 RandPoint
0024 E311     00156         call    Random16
0025 A418     00157         tlwt    _LOW,RandLo       ; only for data capture
0026 AE19     00158         tablwt  _HIGH,0,RandHi    ; using PICMASTER
0027 C024     00159         goto    RandPoint
              00160 ;
0028          00161 GaussPoint
0028 E31E     00162         call    Gauss
0029 A41A     00163         tlwt    _LOW,GaussLo      ; only for data capture
002A AE1B     00164         tablwt  _HIGH,0,GaussHi   ; using PICMASTER
002B C028     00165         goto    GaussPoint
              00166 ;
002C C02C     00167 self    goto    self              ; End Of Test Routines
              00168 ;
002D          00169 loadAB
002D B001     00170         movlw   0x01
002E 0119     00171         movwf   ACCaHI
002F B0FF     00172         movlw   0xff              ; loads ACCa = 01FF
0030 0118     00173         movwf   ACCaLO
              00174 ;
0031 B07F     00175         movlw   0x7f
0032 011B     00176         movwf   ACCbHI
0033 B0FF     00177         movlw   0xFF              ; loads ACCb = 7FFF
0034 011A     00178         movwf   ACCbLO
0035 0002     00179         return
              00180 ;
              00181         PAGE
              00182 ;*****************************************************************
```

```
                00183 ;                 Double Precision Arithmetic Routines
                00184 ;
                00185 ;    Routines : Addition, Subtraction, Multiplication ,Division
                00186 ;               Square Root
                00187 ;
                00188 ;               NOTE :  MODE_FAST must first be set to either
                00189 ;                       TRUE or FALSE
                00190 ;
                00191 ;  MODE_FAST determines the RAM address locations of ACCa thru ACCd
                00192 ;
                00193 ;  If MODE_FAST is set TRUE, data transfers can be done efficiently
                00194 ;  using "MOVFP" & "MOVPF" instructions instead of indirectly moving
                00195 ;  at first to W Reg and then to the desired RAM locations
                00196 ;
                00197 ;         The speed increase using this way of locating ACCa to
                00198 ;         ACCd will result in a saving of about 20 Cycles/filter stage
                00199 ;         In this case ( a 2 stage filter), it is faster by 40 Cycles
                00200 ;
                00201 ;  If due to other constraints, ACCa thru ACCd cannot be set at
                00202 ; address 0x18 to 0x1f, then the user is required to set
                00203 ; MODE_FAST to FALSE
                00204 ;
                00205         PAGE
                00206 ;*****************************************************************
                00207 ;                 Double Precision Addition
                00208 ;
                00209 ;   Addition :  ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
                00210 ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                00211 ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                00212 ;      (c) CALL D_add
                00213 ;      (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
                00214 ;
                00215 ;   Performance :
                00216 ;               Program Memory  :  4 (excluding call & return)
                00217 ;               Clock Cycles    :  4 (excluding call & return)
                00218 ;               W Register      :  Used
                00219 ;               Scratch RAM     :  0
                00220 ;
                00221 ;*****************************************************************;
                00222 ;
0036            00223 D_add
0036 6A18       00224         movfp    ACCaLO,WREG
0037 0F1A       00225         addwf    ACCbLO, F           ;addwf lsb
0038 6A19       00226         movfp    ACCaHI,WREG
0039 111B       00227         addwfc   ACCbHI, F           ;addwf msb with carry
003A 0002       00228         return
                00229 ;
                00230         PAGE
                00231 ;*****************************************************************
                00232 ;                 Double Precision Subtraction
                00233 ;
                00234 ;   Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
                00235 ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                00236 ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                00237 ;      (c) CALL D_sub
                00238 ;      (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
                00239 ;
                00240 ;   Performance :
                00241 ;               Program Memory  :   4  (excluding call & return )
                00242 ;               Clock Cycles    :   4  (excluding call & return )
                00243 ;               W Register      :   Used
                00244 ;               scratch RAM     :   0
                00245 ;*****************************************************************;
                00246 ;
003B            00247 D_sub
003B 6A18       00248         movfp    ACCaLO,WREG
```


© 1997 Microchip Technology Inc.

```
003C 051A     00249         subwf     ACCbLO, F
003D 6A19     00250         movfp     ACCaHI,WREG
003E 031B     00251         subwfb    ACCbHI, F
003F 0002     00252         return
              00253 ;
              00254         PAGE
              00255 ;*****************************************************************
              00256 ;               Function to negate a 16 bit integer
              00257 ;  The two 8 bit integers are assumed to be in 2 consecutive
              00258 ; locations. Before calling this routine, FSR0 should be loaded with
              00259 ; the address of the lower byte.
              00260 ;       Assume that ALUSTA register is set for no autoincrement of
              00261 ; FSR0.
              00262 ;*****************************************************************
              00263 ;
0040          00264 negateAlt
0040 6A00     00265         movfp     INDF0,WREG
0041 8D04     00266         bcf       ALUSTA,FS1
0042 2D00     00267         negw      INDF0, F
0043 8504     00268         bsf       ALUSTA,FS1
0044 6A00     00269         movfp     INDF0,WREG
0045 2900     00270         clrf      INDF0, F
0046 0300     00271         subwfb    INDF0, F
0047 0002     00272         return
              00273 ;
0048          00274 negate
0048 1300     00275         comf      INDF0, F
0049 8D04     00276         bcf       ALUSTA,FS1
004A 1500     00277         incf      INDF0, F
004B 8504     00278         bsf       ALUSTA,FS1
004C 9A04     00279         btfsc     ALUSTA,Z
004D 0700     00280         decf      INDF0, F
004E 1300     00281         comf      INDF0, F
004F 0002     00282         return
              00283 ;
              00284         PAGE
              00285 ;*****************************************************************
              00286 ;                       Double Precision Multiplication
              00287 ;
              00288 ;                ( Optimized for Code : Looped Code )
              00289 ;
              00290 ;   Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCd,ACCc ( 32 bits )
              00291 ;       (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
              00292 ;       (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
              00293 ;       (c) CALL D_mpyS
              00294 ;       (d) The 32 bit result is in location ( ACCdHI,ACCdLO,ACCdHI,ACCdLO )
              00295 ;
              00296 ;   Performance :
              00297 ;             Program Memory  :   21 (UNSIGNED)
              00298 ;                                 52 (SIGNED)
              00299 ;             Clock Cycles    :   242 (UNSIGNED :excluding CALL & RETURN)
              00300 ;                             :   254 (SIGNED :excluding CALL & RETURN)
              00301 ;             Scratch RAM     :   1 (used only if SIGNED arithmetic)
              00302 ;
              00303 ;       Note : The above timing is the worst case timing, when the
              00304 ;                 register ACCb = FFFF. The speed may be improved if
              00305 ;                 the register ACCb contains a number ( out of the two
              00306 ;                 numbers ) with less number of 1s.
              00307 ;
              00308 ;                 Double Precision Multiply ( 16x16 -> 32 )
              00309 ;           ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
              00310 ;  in ACCd ( ACCdHI,ACCdLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
              00311 ;*****************************************************************
              00312 ;
0050          00313 D_mpyS                          ;results in ACCd(16 msb's) and ACCc(16 lsb's)
              00314 ;
```

```
                00315
                00316       #if   SIGNED
                00317          CALL    S_SIGN
                00318       #endif
                00319 ;
0050 2922       00320          clrf     count, F
0051 8422       00321          bsf      count,4                 ; set count = 16
                00322 ;
                00323    #if MODE_FAST
0052 5A20       00324          movpf    ACCbLO,tempLo
0053 5B21       00325          movpf    ACCbHI,tempHi
                00326    #else
                00327          movfp    ACCbLO,WREG
                00328          movwf    tempLo
                00329          movfp    ACCbHI,WREG
                00330          movwf     empHi
                00331    #endif
0054 291F       00332          clrf     ACCdHI, F
0055 291E       00333          clrf     ACCdLO, F
                00334 ;
                00335 ; shift right and addwf 16 times
                00336 ;
0056            00337 mpyLoop
0056 1921       00338          rrcf     tempHi, F
0057 1920       00339          rrcf     tempLo, F
0058 9004       00340          btfss    ALUSTA,C
0059 C05E       00341          goto     NoAdd                   ; LSB is 0, so no need to addwf
005A 6A18       00342          movfp    ACCaLO,WREG
005B 0F1E       00343          addwf    ACCdLO, F               ;addwf lsb
005C 6A19       00344          movfp    ACCaHI,WREG
005D 111F       00345          addwfc   ACCdHI, F               ;addwf msb
005E            00346 NoAdd
005E 191F       00347          rrcf     ACCdHI, F
005F 191E       00348          rrcf     ACCdLO, F
0060 191D       00349          rrcf     ACCcHI, F
0061 191C       00350          rrcf     ACCcLO, F
0062 1722       00351          decfsz   count, F
0063 C056       00352          goto     mpyLoop
                00353 ;
                00354       #if SIGNED
                00355          btfss    sign,MSB
                00356          return
                00357          comf     ACCcLO, F
                00358          incf     ACCcLO, F
                00359          btfsc    ALUSTA,Z
                00360          decf     ACCcHI, F
                00361          comf     ACCcHI, F
                00362          btfsc    ALUSTA,Z
                00363          decf     ACCdLO, F
                00364          comf     ACCdLO, F
                00365          btfsc    ALUSTA,Z
                00366          decf     ACCdHI, F
                00367          comf     ACCdHI, F
                00368          return
                00369       #else
0064 0002       00370          return
                00371       #endif
                00372 ;
                00373 ;  Assemble this section only if Signed Arithmetic Needed
                00374 ;
                00375        #if    SIGNED
                00376 ;
                00377 S_SIGN
                00378          movfp    ACCaHI,WREG
                00379          xorwf    ACCbHI,W
                00380          movwf    sign              ; MSB of sign determines whether signed
```


© 1997 Microchip Technology Inc.

```
00381         btfss    ACCbHI,MSB        ; if MSB set go & negate ACCb
00382         goto     chek_A
00383         comf     ACCbLO, F
00384         incf     ACCbLO, F
00385         btfsc    ALUSTA,Z          ; negate ACCb
00386         decf     ACCbHI, F
00387         comf     ACCbHI, F
00388 ;
00389 chek_A
00390         btfss    ACCaHI,MSB        ; if MSB set go & negate ACCa
00391         return
00392         comf     ACCaLO, F
00393         incf     ACCaLO, F
00394         btfsc    ALUSTA,Z          ; negate ACCa
00395         decf     ACCaHI, F
00396         comf     ACCaHI, F
00397         return
00398 ;
00399      #endif
00400 ;
00401         PAGE
00402 ;*****************************************************************
00403 ;                        Double Precision Multiplication
00404 ;
00405 ;                ( Optimized for Speed : straight Line Code )
00406 ;
00407 ;   Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCd,ACCc ( 32 bits )
00408 ;       (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
00409 ;       (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
00410 ;       (c) CALL D_mpy
00411 ;       (d) The 32 bit result is in location ( ACCdHI,ACCdLO,ACCdHI,ACCdLO )
00412 ;
00413 ;   Performance :
00414 ;             Program Memory  :   179 (UNSIGNED)
00415 ;                                 204 (SIGNED)
00416 ;             Clock Cycles    :   176 (UNSIGNED :excluding CALL & RETURN)
00417 ;                             :   183 (SIGNED :excluding CALL & RETURN)
00418 ;
00419 ;       Note : The above timing is the worst case timing, when the
00420 ;              register ACCb = FFFF. The speed may be improved if
00421 ;              the register ACCb contains a number ( out of the two
00422 ;              numbers ) with less number of 1s.
00423 ;
00424 ;              The performance specs are for Unsigned arithmetic ( i.e,
00425 ;              with "SIGNED equ  FALSE ").
00426 ;
00427 ;              Upon return from subroutine, the input registers
00428 ;
00429 ;
00430 ;*************************************************************
00431 ;               Multiplication Macro
00432 ;*************************************************************
00433 ;
00434 mulMac  MACRO
00435         variable i
00436
00437         variable i = 0
00438
00439         #if SIGNED
00440            variable MUL_LP_CNT = 15
00441         #else
00442            variable MUL_LP_CNT = 16
00443         #endif
00444           .while i < MUL_LP_CNT
00445 ;
00446              .if i < 8
```

```
00447                   btfss       ACCbLO,i        ; test low byte
00448                .else
00449                   btfss       ACCbHI,i-8      ; test high byte
00450                .fi
00451                   goto        NoAdd#v(i)      ; LSB is 0, so no need to addwf
00452                   movfp       ACCaLO,WREG
00453                   addwf       ACCdLO, F       ;addwf lsb
00454                   movfp       ACCaHI,WREG
00455                   addwfc      ACCdHI, F       ;addwf msb
00456 NoAdd#v(i)
00457                   rrcf        ACCdHI, F
00458                   rrcf        ACCdLO, F
00459                   rrcf        ACCcHI, F
00460                   rrcf        ACCcLO, F
00461                   bcf         ALUSTA,C
00462                   variable    i = i+1
00463             .endw
00464           #if SIGNED
00465                   rrcf        ACCdHI, F
00466                   rrcf        ACCdLO, F
00467                   rrcf        ACCcHI, F
00468                   rrcf        ACCcLO, F
00469                   bcf        ALUSTA,C
00470           #endif
00471 ;
00472           ENDM
00473 ;
00474           PAGE
00475 ;*****************************************************************
00476 ;                           Double Precision Negate Macros
00477 ;*****************************************************************
00478 AltNegMac         MACRO     fileRegLo,fileRegHi
00479                             movfp     fileRegLo,WREG
00480                             negw      fileRegLo, F
00481                             movfp     fileRegHi,WREG
00482                             clrf      fileRegHi, F
00483                             subwfb     fileRegHi, F
00484                   ENDM
00485
00486 ;
00487 negMac    MACRO     fileRegLo, fileRegHi
00488           comf      fileRegLo, F        ; negate FileReg ( -FileReg -> FileReg )
00489           incf      fileRegLo, F
00490           btfsc     ALUSTA,Z
00491           decf      fileRegHi, F
00492           comf      fileRegHi, F
00493           ENDM
00494 ;
00495 NegMac32          MACRO     x3,x2,x1,x0
00496           movfp     x3,WREG
00497           negw      x3, F
00498           movfp     x2,WREG
00499           clrf      x2, F
00500           subwfb    x2, F
00501           movfp     x1,WREG
00502           clrf      x1, F
00503           subwfb    x1, F
00504           movfp     x0,WREG
00505           clrf      x0, F
00506           subwfb    x0, F
00507                     ENDM
00508 ;
00509           PAGE
00510 ;*****************************************************************;
00511 ;                 Double Precision Multiply ( 16x16 -> 32 )
00512 ;           ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
```


© 1997 Microchip Technology Inc.

```
                00513 ;  in ACCd ( ACCdHI,ACCdLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
                00514 ;
0065            00515 D_mpyF                            ;results in ACCd(16 msb's) and ACCc(16 lsb's)
                00516 ;
                00517       #if   SIGNED
                00518 ;
                00519           movfp    ACCaHI,WREG
                00520           xorwf    ACCbHI,W
                00521           movwf    sign
                00522           btfss    ACCbHI,MSB         ; if MSB set go & negate ACCb
                00523           goto     chek_A_MSB_MPY
                00524 ;
                00525           negMac   ACCbLO,ACCbHI
                00526 ;
                00527 chek_A_MSB_MPY
                00528           btfss    ACCaHI,MSB         ; if MSB set go & negate ACCa
                00529           goto     continue_MPY
                00530           negMac   ACCaLO,ACCaHI
                00531 ;
                00532       #endif
                00533 ;
0065            00534 continue_MPY
0065 291F       00535           clrf     ACCdHI, F
0066 291E       00536           clrf     ACCdLO, F
0067 8804       00537           bcf      ALUSTA,C
                00538
                00539 ;
                00540 ; use the mulMac macro 16 times
                00541 ;
                00542           mulMac
  0000            M             variable i
                  M
  0000            M             variable i = 0
                  M
                  M             #if SIGNED
                  M                variable MUL_LP_CNT = 15
                  M             #else
  0010            M                variable MUL_LP_CNT = 16
                  M             #endif
                  M               .while i < MUL_LP_CNT
                  M ;
                  M                  .if i < 8
0068 901A         M                     btfss      ACCbLO,i        ; test low byte
                  M                  .else
                  M                     btfss      ACCbHI,i-8      ; test high byte
                  M                  .fi
0069 C06E         M                     goto       NoAdd0          ; LSB is 0, so no need to addwf
006A 6A18         M                     movfp      ACCaLO,WREG
006B 0F1E         M                     addwf      ACCdLO, F       ; addwf lsb
006C 6A19         M                     movfp      ACCaHI,WREG
006D 111F         M                     addwfc     ACCdHI, F       ; addwf msb
006E              M NoAdd0
006E 191F         M                     rrcf       ACCdHI, F
006F 191E         M                     rrcf       ACCdLO, F
0070 191D         M                     rrcf       ACCcHI, F
0071 191C         M                     rrcf       ACCcLO, F
0072 8804         M                     bcf        ALUSTA,C
  0001            M                     variable   i = i+1
                  M ;
                  M                  .if i < 8
0073 911A         M                     btfss      ACCbLO,i        ; test low byte
                  M                  .else
                  M                     btfss      ACCbHI,i-8      ; test high byte
                  M                  .fi
0074 C079         M                     goto       NoAdd1          ; LSB is 0, so no need to addwf
0075 6A18         M                     movfp      ACCaLO,WREG
```


© 1997 Microchip Technology Inc.

```
0076 0F1E        M                addwf     ACCdLO, F          ;addwf lsb
0077 6A19        M                movfp     ACCaHI,WREG
0078 111F        M                addwfc    ACCdHI, F          ;addwf msb
0079             M NoAdd1
0079 191F        M                rrcf      ACCdHI, F
007A 191E        M                rrcf      ACCdLO, F
007B 191D        M                rrcf      ACCcHI, F
007C 191C        M                rrcf      ACCcLO, F
007D 8804        M                bcf       ALUSTA,C
  0002           M                variable  i = i+1
                 M ;
                 M              .if i < 8
007E 921A        M                btfss     ACCbLO,i           ; test low byte
                 M              .else
                 M                btfss     ACCbHI,i-8         ; test high byte
                 M              .fi
007F C084        M                goto      NoAdd2             ; LSB is 0, so no need to addwf
0080 6A18        M                movfp     ACCaLO,WREG
0081 0F1E        M                addwf     ACCdLO, F          ; addwf lsb
0082 6A19        M                movfp     ACCaHI,WREG
0083 111F        M                addwfc    ACCdHI, F          ; addwf msb
0084             M NoAdd2
0084 191F        M                rrcf      ACCdHI, F
0085 191E        M                rrcf      ACCdLO, F
0086 191D        M                rrcf      ACCcHI, F
0087 191C        M                rrcf      ACCcLO, F
0088 8804        M                bcf       ALUSTA,C
  0003           M                variable  i = i+1
                 M ;
                 M              .if i < 8
0089 931A        M                btfss     ACCbLO,i           ; test low byte
                 M              .else
                 M                btfss     ACCbHI,i-8         ; test high byte
                 M              .fi
008A C08F        M                goto      NoAdd3               ; LSB is 0, so no need to addwf
008B 6A18        M                movfp     ACCaLO,WREG
008C 0F1E        M                addwf     ACCdLO, F          ; addwf lsb
008D 6A19        M                movfp     ACCaHI,WREG
008E 111F        M                addwfc    ACCdHI, F          ; addwf msb
008F             M NoAdd3
008F 191F        M                rrcf      ACCdHI, F
0090 191E        M                rrcf      ACCdLO, F
0091 191D        M                rrcf      ACCcHI, F
0092 191C        M                rrcf      ACCcLO, F
0093 8804        M                bcf       ALUSTA,C
  0004           M                variable  i = i+1
                 M ;
                 M              .if i < 8
0094 941A        M                btfss     ACCbLO,i           ; test low byte
                 M              .else
                 M                btfss     ACCbHI,i-8         ; test high byte
                 M              .fi
0095 C09A        M                goto      NoAdd4             ; LSB is 0, so no need to addwf
0096 6A18        M                movfp     ACCaLO,WREG
0097 0F1E        M                addwf     ACCdLO, F          ; addwf lsb
0098 6A19        M                movfp     ACCaHI,WREG
0099 111F        M                addwfc    ACCdHI, F          ; addwf msb
009A             M NoAdd4
009A 191F        M                rrcf      ACCdHI, F
009B 191E        M                rrcf      ACCdLO, F
009C 191D        M                rrcf      ACCcHI, F
009D 191C        M                rrcf      ACCcLO, F
009E 8804        M                bcf       ALUSTA,C
  0005           M                variable  i = i+1
                 M ;
                 M              .if i < 8
```


© 1997 Microchip Technology Inc.

```
009F 951A          M                btfss     ACCbLO,i        ; test low byte
                   M              .else
                   M                btfss     ACCbHI,i-8      ; test high byte
                   M              .fi
00A0 C0A5          M                goto      NoAdd5          ; LSB is 0, so no need to addwf
00A1 6A18          M                movfp     ACCaLO,WREG
00A2 0F1E          M                addwf     ACCdLO, F       ; addwf lsb
00A3 6A19          M                movfp     ACCaHI,WREG
00A4 111F          M                addwfc    ACCdHI, F       ; addwf msb
00A5               M NoAdd5
00A5 191F          M                rrcf      ACCdHI, F
00A6 191E          M                rrcf      ACCdLO, F
00A7 191D          M                rrcf      ACCcHI, F
00A8 191C          M                rrcf      ACCcLO, F
00A9 8804          M                bcf       ALUSTA,C
  0006             M                variable  i = i+1
                   M ;
                   M              .if i < 8
00AA 961A          M                btfss     ACCbLO,i        ; test low byte
                   M              .else
                   M                btfss     ACCbHI,i-8      ; test high byte
                   M              .fi
00AB C0B0          M                goto      NoAdd6          ; LSB is 0, so no need to addwf
00AC 6A18          M                movfp     ACCaLO,WREG
00AD 0F1E          M                addwf     ACCdLO, F       ; addwf lsb
00AE 6A19          M                movfp     ACCaHI,WREG
00AF 111F          M                addwfc    ACCdHI, F       ; addwf msb
00B0               M NoAdd6
00B0 191F          M                rrcf      ACCdHI, F
00B1 191E          M                rrcf      ACCdLO, F
00B2 191D          M                rrcf      ACCcHI, F
00B3 191C          M                rrcf      ACCcLO, F
00B4 8804          M                bcf       ALUSTA,C
  0007             M                variable  i = i+1
                   M ;
                   M              .if i < 8
00B5 971A          M                btfss     ACCbLO,i        ; test low byte
                   M              .else
                   M                btfss     ACCbHI,i-8      ; test high byte
                   M              .fi
00B6 C0BB          M                goto      NoAdd7          ; LSB is 0, so no need to addwf
00B7 6A18          M                movfp     ACCaLO,WREG
00B8 0F1E          M                addwf     ACCdLO, F       ; addwf lsb
00B9 6A19          M                movfp     ACCaHI,WREG
00BA 111F          M                addwfc    ACCdHI, F       ; addwf msb
00BB               M NoAdd7
00BB 191F          M                rrcf      ACCdHI, F
00BC 191E          M                rrcf      ACCdLO, F
00BD 191D          M                rrcf      ACCcHI, F
00BE 191C          M                rrcf      ACCcLO, F
00BF 8804          M                bcf       ALUSTA,C
  0008             M                variable  i = i+1
                   M ;
                   M              .if i < 8
                   M                btfss     ACCbLO,i        ; test low byte
                   M              .else
00C0 901B          M                btfss     ACCbHI,i-8      ; test high byte
                   M              .fi
00C1 C0C6          M                goto      NoAdd8          ; LSB is 0, so no need to addwf
00C2 6A18          M                movfp     ACCaLO,WREG
00C3 0F1E          M                addwf     ACCdLO, F       ; addwf lsb
00C4 6A19          M                movfp     ACCaHI,WREG
00C5 111F          M                addwfc    ACCdHI, F       ; addwf msb
00C6               M NoAdd8
00C6 191F          M                rrcf      ACCdHI, F
00C7 191E          M                rrcf      ACCdLO, F
```


© 1997 Microchip Technology Inc.

```
00C8 191D       M                rrcf       ACCcHI, F
00C9 191C       M                rrcf       ACCcLO, F
00CA 8804       M                bcf        ALUSTA,C
  0009          M                variable   i = i+1
                M ;
                M             .if i < 8
                M                btfss      ACCbLO,i        ; test low byte
                M             .else
00CB 911B       M                btfss      ACCbHI,i-8      ; test high byte
                M             .fi
00CC C0D1       M                goto       NoAdd9          ; LSB is 0, so no need to addwf
00CD 6A18       M                movfp      ACCaLO,WREG
00CE 0F1E       M                addwf      ACCdLO, F       ; addwf lsb
00CF 6A19       M                movfp      ACCaHI,WREG
00D0 111F       M                addwfc     ACCdHI, F       ; addwf msb
00D1            M NoAdd9
00D1 191F       M                rrcf       ACCdHI, F
00D2 191E       M                rrcf       ACCdLO, F
00D3 191D       M                rrcf       ACCcHI, F
00D4 191C       M                rrcf       ACCcLO, F
00D5 8804       M                bcf        ALUSTA,C
  000A          M                variable   i = i+1
                M ;
                M             .if i < 8
                M                btfss      ACCbLO,i        ; test low byte
                M             .else
00D6 921B       M                btfss      ACCbHI,i-8      ; test high byte
                M             .fi
00D7 C0DC       M                goto       NoAdd10         ; LSB is 0, so no need to addwf
00D8 6A18       M                movfp      ACCaLO,WREG
00D9 0F1E       M                addwf      ACCdLO, F       ; addwf lsb
00DA 6A19       M                movfp      ACCaHI,WREG
00DB 111F       M                addwfc     ACCdHI, F       ; addwf msb
00DC            M NoAdd10
00DC 191F       M                rrcf       ACCdHI, F
00DD 191E       M                rrcf       ACCdLO, F
00DE 191D       M                rrcf       ACCcHI, F
00DF 191C       M                rrcf       ACCcLO, F
00E0 8804       M                bcf        ALUSTA,C
  000B          M                variable   i = i+1
                M ;
                M             .if i < 8
                M                btfss      ACCbLO,i        ; test low byte
                M             .else
00E1 931B       M                btfss      ACCbHI,i-8      ; test high byte
                M             .fi
00E2 C0E7       M                goto       NoAdd11         ; LSB is 0, so no need to addwf
00E3 6A18       M                movfp      ACCaLO,WREG
00E4 0F1E       M                addwf      ACCdLO, F       ; addwf lsb
00E5 6A19       M                movfp      ACCaHI,WREG
00E6 111F       M                addwfc     ACCdHI, F       ; addwf msb
00E7            M NoAdd11
00E7 191F       M                rrcf       ACCdHI, F
00E8 191E       M                rrcf       ACCdLO, F
00E9 191D       M                rrcf       ACCcHI, F
00EA 191C       M                rrcf       ACCcLO, F
00EB 8804       M                bcf        ALUSTA,C
  000C          M                variable   i = i+1
                M ;
                M             .if i < 8
                M                btfss      ACCbLO,i        ; test low byte
                M             .else
00EC 941B       M                btfss      ACCbHI,i-8      ; test high byte
                M             .fi
00ED C0F2       M                goto       NoAdd12         ; LSB is 0, so no need to addwf
00EE 6A18       M                movfp      ACCaLO,WREG
```


© 1997 Microchip Technology Inc.

```
00EF 0F1E       M                addwf      ACCdLO, F          ;addwf lsb
00F0 6A19       M                movfp      ACCaHI,WREG
00F1 111F       M                addwfc     ACCdHI, F          ;addwf msb
00F2            M NoAdd12
00F2 191F       M                rrcf       ACCdHI, F
00F3 191E       M                rrcf       ACCdLO, F
00F4 191D       M                rrcf       ACCcHI, F
00F5 191C       M                rrcf       ACCcLO, F
00F6 8804       M                bcf        ALUSTA,C
  000D          M                variable   i = i+1
                M ;
                M             .if i < 8
                M                btfss      ACCbLO,i        ; test low byte
                M             .else
00F7 951B       M                btfss      ACCbHI,i-8      ; test high byte
                M             .fi
00F8 C0FD       M                goto       NoAdd13         ; LSB is 0, so no need to addwf
00F9 6A18       M                movfp      ACCaLO,WREG
00FA 0F1E       M                addwf      ACCdLO, F       ; addwf lsb
00FB 6A19       M                movfp      ACCaHI,WREG
00FC 111F       M                addwfc     ACCdHI, F       ; addwf msb
00FD            M NoAdd13
00FD 191F       M                rrcf       ACCdHI, F
00FE 191E       M                rrcf       ACCdLO, F
00FF 191D       M                rrcf       ACCcHI, F
0100 191C       M                rrcf       ACCcLO, F
0101 8804       M                bcf        ALUSTA,C
  000E          M                variable   i = i+1
                M ;
                M             .if i < 8
                M                btfss      ACCbLO,i        ; test low byte
                M             .else
0102 961B       M                btfss      ACCbHI,i-8      ; test high byte
                M             .fi
0103 C108       M                goto       NoAdd14         ; LSB is 0, so no need to addwf
0104 6A18       M                movfp      ACCaLO,WREG
0105 0F1E       M                addwf      ACCdLO, F       ; addwf lsb
0106 6A19       M                movfp      ACCaHI,WREG
0107 111F       M                addwfc     ACCdHI, F       ; addwf msb
0108            M NoAdd14
0108 191F       M                rrcf       ACCdHI, F
0109 191E       M                rrcf       ACCdLO, F
010A 191D       M                rrcf       ACCcHI, F
010B 191C       M                rrcf       ACCcLO, F
010C 8804       M                bcf        ALUSTA,C
  000F          M                variable   i = i+1
                M ;
                M             .if i < 8
                M                btfss      ACCbLO,i        ; test low byte
                M             .else
010D 971B       M                btfss      ACCbHI,i-8      ; test high byte
                M             .fi
010E C113       M                goto       NoAdd15         ; LSB is 0, so no need to addwf
010F 6A18       M                movfp      ACCaLO,WREG
0110 0F1E       M                addwf      ACCdLO, F       ; addwf lsb
0111 6A19       M                movfp      ACCaHI,WREG
0112 111F       M                addwfc     ACCdHI, F        ; addwf msb
0113            M NoAdd15
0113 191F       M                rrcf       ACCdHI, F
0114 191E       M                rrcf       ACCdLO, F
0115 191D       M                rrcf       ACCcHI, F
0116 191C       M                rrcf       ACCcLO, F
0117 8804       M                bcf        ALUSTA,C
  0010          M                variable   i = i+1
                M          .endw
                M       #if SIGNED
```


© 1997 Microchip Technology Inc.

```
                M                rrcf       ACCdHI, F
                M                rrcf       ACCdLO, F
                M                rrcf       ACCcHI, F
                M                rrcf       ACCcLO, F
                M                bcf        ALUSTA,C
                M        #endif
                M ;
             00543 ;
             00544     #if SIGNED
             00545         btfss   sign,MSB        ; negate (ACCc,ACCd)
             00546         return
             00547         NegMac32 ACCcHI,ACCcLO,ACCdHI, ACCdLO
             00548         return
             00549     #else
0118 0002    00550         return
             00551     #endif
             00552 ;
             00553         PAGE
             00554 ;*****************************************************************
             00555 ;                       Double Precision Division
             00556 ;
             00557 ;               ( Optimized for Code : Looped Code )
             00558 ;
             00559 ;*****************************************************************;
             00560 ;   Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
             00561 ;                                              Remainder in ACCc (16 bits)
             00562 ;       (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
             00563 ;       (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
             00564 ;       (c) CALL D_div
             00565 ;       (d) The 16 bit result is in location ACCbHI & ACCbLO
             00566 ;       (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
             00567 ;
             00568 ;   Performance :
             00569 ;            Program Memory     :   31 (UNSIGNED)
             00570 ;                                   39 (SIGNED)
             00571 ;               Clock Cycles    :   300 (UNSIGNED : excluding CALL & RETURN)
             00572 ;                               :   312 (SIGNED : excluding CALL & RETURN)
             00573 ;
             00574 ;       NOTE :
             00575 ;               The performance specs are for Unsigned arithmetic ( i.e,
             00576 ;               with "SIGNED equ  FALSE ").
             00577 ;
             00578 ;*****************************************************************
             00579 ;       Double Precision Divide ( 16/16 -> 16 )
             00580 ;
             00581 ;         ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
             00582 ; with Quotiont in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc (ACCcHI,ACCcLO).
             00583 ;
             00584 ;               B/A = (Q) + (R)/A
             00585 ;       or      B = A*Q + R
             00586 ;
             00587 ;                       where   B :  Numerator
             00588 ;                               A :  Denominator
             00589 ;                               Q :  Quotiont (Integer Result)
             00590 ;                               R :  Remainder
             00591 ;
             00592 ;       Note :  Check for ZERO Denominator or Numerator is not performed
             00593 ;               A ZERO Denominator will produce incorrect results
             00594 ;
             00595 ;       SIGNED Arithmetic :
             00596 ;                       In case of signed arithmetic, if either
             00597 ;  numerator or denominator is negative, then both Q & R are
             00598 ;  represented as negative numbers
             00599 ;               -(B/A) = -(Q) + (-R)/A
             00600 ;       or      -B = (-Q)*A + (-R)
             00601 ;
```


© 1997 Microchip Technology Inc.

```
                00602 ;*****************************************************************
                00603 ;
0119            00604 D_divS
                00605 ;
0119 8404       00606           bsf     ALUSTA,FS0
011A 8504       00607           bsf     ALUSTA,FS1              ; set no auto-incrment for fsr0
                00608
                00609       #if    SIGNED
                00610           CALL    S_SIGN
                00611       #endif
                00612 ;
011B 2922       00613           clrf    count, F
011C 8422       00614           bsf     count,4                 ; set count = 16
011D 291D       00615           clrf    ACCcHI, F
011E 291C       00616           clrf    ACCcLO, F
011F 291E       00617           clrf    ACCdLO, F
0120 291F       00618           clrf    ACCdHI, F
                00619 ;
                00620 ;  Looped code
                00621 ;
0121            00622 divLoop
0121 8804       00623           bcf     ALUSTA,C
0122 1B1A       00624           rlcf    ACCbLO, F
0123 1B1B       00625           rlcf    ACCbHI, F
0124 1B1C       00626           rlcf    ACCcLO, F
0125 1B1D       00627           rlcf    ACCcHI, F
0126 6A19       00628           movfp   ACCaHI,WREG
0127 041D       00629           subwf   ACCcHI,W                ; check if a>c
0128 9204       00630           btfss   ALUSTA,Z
0129 C12C       00631           goto    notz
012A 6A18       00632           movfp   ACCaLO,WREG
012B 041C       00633           subwf   ACCcLO,W                ; if msb equal then check lsb
012C            00634 notz
012C 9004       00635           btfss   ALUSTA,C                ; carry set if c>a
012D C133       00636           goto    nosub                   ; if c < a
012E            00637 subca
012E 6A18       00638           movfp   ACCaLO,WREG             ; c-a into c
012F 051C       00639           subwf   ACCcLO, F
0130 6A19       00640           movfp   ACCaHI,WREG
0131 031D       00641           subwfb  ACCcHI, F
0132 8004       00642           bsf     ALUSTA,C                ; shift a 1 into d (result)
0133            00643 nosub
0133 1B1E       00644           rlcf    ACCdLO, F
0134 1B1F       00645           rlcf    ACCdHI, F
0135 1722       00646           decfsz  count, F
0136 C121       00647           goto    divLoop
                00648
                00649 ;
                00650       #if SIGNED
                00651           btfss   sign,MSB
                00652           return
                00653           movlw   ACCcLO
                00654           movwf   fsr0
                00655           call    negate
                00656           movlw   ACCdLO
                00657           movwf   fsr0
                00658           call    negate
                00659           return
                00660       #else
0137 0002       00661           return
                00662       #endif
                00663 ;
                00664           PAGE
                00665 ;*****************************************************************
                00666 ;                       Double Precision Division
                00667 ;
```


© 1997 Microchip Technology Inc.

```
00668 ;               ( Optimized for Speed : straight Line Code )
00669 ;
00670 ;*****************************************************************;
00671 ;   Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
00672 ;                                              Remainder in ACCc (16 bits)
00673 ;
00674 ;       (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
00675 ;       (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
00676 ;       (c) CALL D_div
00677 ;       (d) The 16 bit result is in location ACCbHI & ACCbLO
00678 ;       (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
00679 ;
00680 ;               B/A = (Q) + (R)/A
00681 ;       or      B = A*Q + R
00682 ;
00683 ;                       where   B :  Numerator
00684 ;                               A :  Denominator
00685 ;                               Q :  Quotiont (Integer Result)
00686 ;                               R :  Remainder
00687 ;
00688 ;       Note :  Check for ZERO Denominator or Numerator is not performed
00689 ;               A ZERO Denominator will produce incorrect results
00690 ;
00691 ;       SIGNED Arithmetic :
00692 ;                       In case of signed arithmetic, if either
00693 ;  numerator or denominator is negative, then both Q & R are
00694 ;  represented as negative numbers
00695 ;               -(B/A) = -(Q) + (-R)/A
00696 ;       or      -B = (-Q)*A + (-R)
00697 ;
00698 ;   Performance :
00699 ;            Program Memory     :   325 (UNSIGNED)
00700 ;                               :   354 (SIGNED)
00701 ;               Clock Cycles    :   250 (UNSIGNED : excluding CALL & RETURN)
00702 ;                               :   260 (SIGNED : excluding CALL & RETURN)
00703 ;
00704 ;*****************************************************************;
00705 ;       division macro
00706 ;
00707 divMac  MACRO
00708         variable i
00709
00710        variable i = 0
00711       .while i < 16
00712 ;
00713                 bcf     ALUSTA,C
00714                 rlcf    ACCbLO, F
00715                 rlcf    ACCbHI, F
00716                 rlcf    ACCcLO, F
00717                 rlcf    ACCcHI, F
00718                 movfp   ACCaHI,WREG
00719                 subwf   ACCcHI,W        ; check if a>c
00720                 btfss   ALUSTA,Z
00721                 goto    notz#v(i)
00722                 movfp   ACCaLO,WREG
00723                 subwf   ACCcLO,W        ; if msb equal then check lsb
00724 notz#v(i)       btfss   ALUSTA,C        ; carry set if c>a
00725                 goto    nosub#v(i)      ; if c < a
00726 subca#v(i)      movfp   ACCaLO,WREG     ; c-a into c
00727                 subwf   ACCcLO, F
00728                 movfp   ACCaHI,WREG
00729                 subwfb  ACCcHI, F
00730                 bsf     ALUSTA,C        ; shift a 1 into d (result)
00731 nosub#v(i)      rlcf    ACCdLO, F
00732                 rlcf    ACCdHI, F
00733        variable i=i+1
```


© 1997 Microchip Technology Inc.

```
                00734        .endw
                00735 ;
                00736          ENDM
                00737 ;
                00738          PAGE
                00739 ;*****************************************************************
                00740 ;        Double Precision Divide ( 16/16 -> 16 )
                00741 ;
                00742 ;          ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
                00743 ; with Quotiont in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc (ACCcHI,ACCcLO).
                00744 ;
                00745 ;    NOTE  :  Before calling this routine, the user should make sure that
                00746 ;             the Numerator(ACCb) is greater than Denominator(ACCa). If
                00747 ;             the case is not true, the user should scale either Numerator
                00748 ;             or Denominator or both such that Numerator is greater than
                00749 ;             the Denominator.
                00750 ;
                00751 ;
                00752 ;*****************************************************************
                00753 ;
0138            00754 D_divF
                00755 ;
                00756       #if   SIGNED
                00757          movfp      ACCaHI,WREG
                00758          xorwf      ACCbHI,W
                00759          movwf      sign
                00760          btfss      ACCbHI,MSB          ; if MSB set go & negate ACCb
                00761          goto       chek_A_MSB_DIV
                00762 ;
                00763          negMac     ACCbLO,ACCbHI
                00764 ;
                00765 chek_A_MSB_DIV
                00766          btfss      ACCaHI,MSB          ; if MSB set go & negate ACCa
                00767          goto       continue_DIV
                00768          negMac     ACCaLO,ACCaHI
                00769
                00770       #endif
                00771 ;
0138            00772 continue_DIV
0138 291D       00773          clrf       ACCcHI, F
0139 291C       00774          clrf       ACCcLO, F
013A 291E       00775          clrf       ACCdLO, F
013B 291F       00776          clrf       ACCdHI, F
                00777
                00778 ;
                00779 ; straight line code : using the macro divMac
                00780 ;
                00781          divMac
  0000              M          variable i
                    M
  0000              M         variable i = 0
                    M        .while i < 16
                    M ;
013C 8804           M                  bcf      ALUSTA,C
013D 1B1A           M                  rlcf     ACCbLO, F
013E 1B1B           M                  rlcf     ACCbHI, F
013F 1B1C           M                  rlcf     ACCcLO, F
0140 1B1D           M                  rlcf     ACCcHI, F
0141 6A19           M                  movfp    ACCaHI,WREG
0142 041D           M                  subwf    ACCcHI,W          ; check if a>c
0143 9204           M                  btfss    ALUSTA,Z
0144 C147           M                  goto     notz0
0145 6A18           M                  movfp    ACCaLO,WREG
0146 041C           M                  subwf    ACCcLO,W          ; if msb equal then check lsb
0147 9004           M notz0            btfss    ALUSTA,C          ; carry set if c>a
0148 C14E           M                  goto     nosub0            ; if c < a
```

```
0149 6A18          M subca0          movfp     ACCaLO,WREG          ; c-a into c
014A 051C          M                 subwf     ACCcLO, F
014B 6A19          M                 movfp     ACCaHI,WREG
014C 031D          M                 subwfb    ACCcHI, F
014D 8004          M                 bsf       ALUSTA,C             ; shift a 1 into d (result)
014E 1B1E          M nosub0          rlcf      ACCdLO, F
014F 1B1F          M                 rlcf      ACCdHI, F
  0001             M         variable i=i+1
                   M ;
0150 8804          M                 bcf       ALUSTA,C
0151 1B1A          M                 rlcf      ACCbLO, F
0152 1B1B          M                 rlcf      ACCbHI, F
0153 1B1C          M                 rlcf      ACCcLO, F
0154 1B1D          M                 rlcf      ACCcHI, F
0155 6A19          M                 movfp     ACCaHI,WREG
0156 041D          M                 subwf     ACCcHI,W             ; check if a>c
0157 9204          M                 btfss     ALUSTA,Z
0158 C15B          M                 goto      notz1
0159 6A18          M                 movfp     ACCaLO,WREG
015A 041C          M                 subwf     ACCcLO,W             ; if msb equal then check lsb
015B 9004          M notz1           btfss     ALUSTA,C             ; carry set if c>a
015C C162          M                 goto      nosub1               ; if c < a
015D 6A18          M subca1          movfp     ACCaLO,WREG          ; c-a into c
015E 051C          M                 subwf     ACCcLO, F
015F 6A19          M                 movfp     ACCaHI,WREG
0160 031D          M                 subwfb    ACCcHI, F
0161 8004          M                 bsf       ALUSTA,C             ; shift a 1 into d (result)
0162 1B1E          M nosub1          rlcf      ACCdLO, F
0163 1B1F          M                 rlcf      ACCdHI, F
  0002             M         variable i=i+1
                   M ;
0164 8804          M                 bcf       ALUSTA,C
0165 1B1A          M                 rlcf      ACCbLO, F
0166 1B1B          M                 rlcf      ACCbHI, F
0167 1B1C          M                 rlcf      ACCcLO, F
0168 1B1D          M                 rlcf      ACCcHI, F
0169 6A19          M                 movfp     ACCaHI,WREG
016A 041D          M                 subwf     ACCcHI,W             ; check if a>c
016B 9204          M                 btfss     ALUSTA,Z
016C C16F          M                 goto      notz2
016D 6A18          M                 movfp     ACCaLO,WREG
016E 041C          M                 subwf     ACCcLO,W             ; if msb equal then check lsb
016F 9004          M notz2           btfss     ALUSTA,C             ; carry set if c>a
0170 C176          M                 goto      nosub2               ; if c < a
0171 6A18          M subca2          movfp     ACCaLO,WREG          ; c-a into c
0172 051C          M                 subwf     ACCcLO, F
0173 6A19          M                 movfp     ACCaHI,WREG
0174 031D          M                 subwfb    ACCcHI, F
0175 8004          M                 bsf       ALUSTA,C             ; shift a 1 into d (result)
0176 1B1E          M nosub2          rlcf      ACCdLO, F
0177 1B1F          M                 rlcf      ACCdHI, F
  0003             M         variable i=i+1
                   M ;
0178 8804          M                 bcf       ALUSTA,C
0179 1B1A          M                 rlcf      ACCbLO, F
017A 1B1B          M                 rlcf      ACCbHI, F
017B 1B1C          M                 rlcf      ACCcLO, F
017C 1B1D          M                 rlcf      ACCcHI, F
017D 6A19          M                 movfp     ACCaHI,WREG
017E 041D          M                 subwf     ACCcHI,W             ; check if a>c
017F 9204          M                 btfss     ALUSTA,Z
0180 C183          M                 goto      notz3
0181 6A18          M                 movfp     ACCaLO,WREG
0182 041C          M                 subwf     ACCcLO,W             ; if msb equal then check lsb
0183 9004          M notz3           btfss     ALUSTA,C             ; carry set if c>a
0184 C18A          M                 goto      nosub3               ; if c < a
```


© 1997 Microchip Technology Inc.

```
0185 6A18          M subca3          movfp     ACCaLO,WREG          ; c-a into c
0186 051C          M                 subwf     ACCcLO, F
0187 6A19          M                 movfp     ACCaHI,WREG
0188 031D          M                 subwfb    ACCcHI, F
0189 8004          M                 bsf       ALUSTA,C             ; shift a 1 into d (result)
018A 1B1E          M nosub3          rlcf      ACCdLO, F
018B 1B1F          M                 rlcf      ACCdHI, F
  0004             M         variable i=i+1
                   M ;
018C 8804          M                 bcf       ALUSTA,C
018D 1B1A          M                 rlcf      ACCbLO, F
018E 1B1B          M                 rlcf      ACCbHI, F
018F 1B1C          M                 rlcf      ACCcLO, F
0190 1B1D          M                 rlcf      ACCcHI, F
0191 6A19          M                 movfp     ACCaHI,WREG
0192 041D          M                 subwf     ACCcHI,W             ; check if a>c
0193 9204          M                 btfss     ALUSTA,Z
0194 C197          M                 goto      notz4
0195 6A18          M                 movfp     ACCaLO,WREG
0196 041C          M                 subwf     ACCcLO,W             ; if msb equal then check lsb
0197 9004          M notz4           btfss     ALUSTA,C             ; carry set if c>a
0198 C19E          M                 goto      nosub4               ; if c < a
0199 6A18          M subca4          movfp     ACCaLO,WREG          ; c-a into c
019A 051C          M                 subwf     ACCcLO, F
019B 6A19          M                 movfp     ACCaHI,WREG
019C 031D          M                 subwfb    ACCcHI, F
019D 8004          M                 bsf       ALUSTA,C             ; shift a 1 into d (result)
019E 1B1E          M nosub4          rlcf      ACCdLO, F
019F 1B1F          M                 rlcf      ACCdHI, F
  0005             M         variable i=i+1
                   M ;
01A0 8804          M                 bcf       ALUSTA,C
01A1 1B1A          M                 rlcf      ACCbLO, F
01A2 1B1B          M                 rlcf      ACCbHI, F
01A3 1B1C          M                 rlcf      ACCcLO, F
01A4 1B1D          M                 rlcf      ACCcHI, F
01A5 6A19          M                 movfp     ACCaHI,WREG
01A6 041D          M                 subwf     ACCcHI,W             ; check if a>c
01A7 9204          M                 btfss     ALUSTA,Z
01A8 C1AB          M                 goto      notz5
01A9 6A18          M                 movfp     ACCaLO,WREG
01AA 041C          M                 subwf     ACCcLO,W             ; if msb equal then check lsb
01AB 9004          M notz5           btfss     ALUSTA,C             ; carry set if c>a
01AC C1B2          M                 goto      nosub5               ; if c < a
01AD 6A18          M subca5          movfp     ACCaLO,WREG          ; c-a into c
01AE 051C          M                 subwf     ACCcLO, F
01AF 6A19          M                 movfp     ACCaHI,WREG
01B0 031D          M                 subwfb    ACCcHI, F
01B1 8004          M                 bsf       ALUSTA,C             ; shift a 1 into d (result)
01B2 1B1E          M nosub5          rlcf      ACCdLO, F
01B3 1B1F          M                 rlcf      ACCdHI, F
  0006             M         variable i=i+1
                   M ;
01B4 8804          M                 bcf       ALUSTA,C
01B5 1B1A          M                 rlcf      ACCbLO, F
01B6 1B1B          M                 rlcf      ACCbHI, F
01B7 1B1C          M                 rlcf      ACCcLO, F
01B8 1B1D          M                 rlcf      ACCcHI, F
01B9 6A19          M                 movfp     ACCaHI,WREG
01BA 041D          M                 subwf     ACCcHI,W             ; check if a>c
01BB 9204          M                 btfss     ALUSTA,Z
01BC C1BF          M                 goto      notz6
01BD 6A18          M                 movfp     ACCaLO,WREG
01BE 041C          M                 subwf     ACCcLO,W             ; if msb equal then check lsb
01BF 9004          M notz6           btfss     ALUSTA,C             ; carry set if c>a
01C0 C1C6          M                 goto      nosub6               ; if c < a
```


© 1997 Microchip Technology Inc.

```
01C1 6A18          M subca6          movfp     ACCaLO,WREG         ; c-a into c
01C2 051C          M                 subwf     ACCcLO, F
01C3 6A19          M                 movfp     ACCaHI,WREG
01C4 031D          M                 subwfb    ACCcHI, F
01C5 8004          M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
01C6 1B1E          M nosub6          rlcf      ACCdLO, F
01C7 1B1F          M                 rlcf      ACCdHI, F
  0007             M        variable i=i+1
                   M ;
01C8 8804          M                 bcf       ALUSTA,C
01C9 1B1A          M                 rlcf      ACCbLO, F
01CA 1B1B          M                 rlcf      ACCbHI, F
01CB 1B1C          M                 rlcf      ACCcLO, F
01CC 1B1D          M                 rlcf      ACCcHI, F
01CD 6A19          M                 movfp     ACCaHI,WREG
01CE 041D          M                 subwf     ACCcHI,W            ; check if a>c
01CF 9204          M                 btfss     ALUSTA,Z
01D0 C1D3          M                 goto      notz7
01D1 6A18          M                 movfp     ACCaLO,WREG
01D2 041C          M                 subwf     ACCcLO,W            ; if msb equal then check lsb
01D3 9004          M notz7           btfss     ALUSTA,C            ; carry set if c>a
01D4 C1DA          M                 goto      nosub7              ; if c < a
01D5 6A18          M subca7          movfp     ACCaLO,WREG         ; c-a into c
01D6 051C          M                 subwf     ACCcLO, F
01D7 6A19          M                 movfp     ACCaHI,WREG
01D8 031D          M                 subwfb    ACCcHI, F
01D9 8004          M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
01DA 1B1E          M nosub7          rlcf      ACCdLO, F
01DB 1B1F          M                 rlcf      ACCdHI, F
  0008             M        variable i=i+1
                   M ;
01DC 8804          M                 bcf       ALUSTA,C
01DD 1B1A          M                 rlcf      ACCbLO, F
01DE 1B1B          M                 rlcf      ACCbHI, F
01DF 1B1C          M                 rlcf      ACCcLO, F
01E0 1B1D          M                 rlcf      ACCcHI, F
01E1 6A19          M                 movfp     ACCaHI,WREG
01E2 041D          M                 subwf     ACCcHI,W            ; check if a>c
01E3 9204          M                 btfss     ALUSTA,Z
01E4 C1E7          M                 goto      notz8
01E5 6A18          M                 movfp     ACCaLO,WREG
01E6 041C          M                 subwf     ACCcLO,W            ; if msb equal then check lsb
01E7 9004          M notz8           btfss     ALUSTA,C            ; carry set if c>a
01E8 C1EE          M                 goto      nosub8              ; if c < a
01E9 6A18          M subca8          movfp     ACCaLO,WREG         ; c-a into c
01EA 051C          M                 subwf     ACCcLO, F
01EB 6A19          M                 movfp     ACCaHI,WREG
01EC 031D          M                 subwfb    ACCcHI, F
01ED 8004          M                 bsf       ALUSTA,C            ; shift a 1 into d (result)
01EE 1B1E          M nosub8          rlcf      ACCdLO, F
01EF 1B1F          M                 rlcf      ACCdHI, F
  0009             M        variable i=i+1
                   M ;
01F0 8804          M                 bcf       ALUSTA,C
01F1 1B1A          M                 rlcf      ACCbLO, F
01F2 1B1B          M                 rlcf      ACCbHI, F
01F3 1B1C          M                 rlcf      ACCcLO, F
01F4 1B1D          M                 rlcf      ACCcHI, F
01F5 6A19          M                 movfp     ACCaHI,WREG
01F6 041D          M                 subwf     ACCcHI,W            ; check if a>c
01F7 9204          M                 btfss     ALUSTA,Z
01F8 C1FB          M                 goto      notz9
01F9 6A18          M                 movfp     ACCaLO,WREG
01FA 041C          M                 subwf     ACCcLO,W            ; if msb equal then check lsb
01FB 9004          M notz9           btfss     ALUSTA,C            ; carry set if c>a
01FC C202          M                 goto      nosub9              ; if c < a
```


© 1997 Microchip Technology Inc.

```
01FD 6A18          M subca9          movfp     ACCaLO,WREG          ; c-a into c
01FE 051C          M                 subwf     ACCcLO, F
01FF 6A19          M                 movfp     ACCaHI,WREG
0200 031D          M                 subwfb    ACCcHI, F
0201 8004          M                 bsf       ALUSTA,C             ; shift a 1 into d (result)
0202 1B1E          M nosub9          rlcf      ACCdLO, F
0203 1B1F          M                 rlcf      ACCdHI, F
  000A             M         variable i=i+1
                   M ;
0204 8804          M                 bcf       ALUSTA,C
0205 1B1A          M                 rlcf      ACCbLO, F
0206 1B1B          M                 rlcf      ACCbHI, F
0207 1B1C          M                 rlcf      ACCcLO, F
0208 1B1D          M                 rlcf      ACCcHI, F
0209 6A19          M                 movfp     ACCaHI,WREG
020A 041D          M                 subwf     ACCcHI,W             ; check if a>c
020B 9204          M                 btfss     ALUSTA,Z
020C C20F          M                 goto      notz10
020D 6A18          M                 movfp     ACCaLO,WREG
020E 041C          M                 subwf     ACCcLO,W             ; if msb equal then check lsb
020F 9004          M notz10          btfss     ALUSTA,C             ; carry set if c>a
0210 C216          M                 goto      nosub10              ; if c < a
0211 6A18          M subca10         movfp     ACCaLO,WREG          ; c-a into c
0212 051C          M                 subwf     ACCcLO, F
0213 6A19          M                 movfp     ACCaHI,WREG
0214 031D          M                 subwfb    ACCcHI, F
0215 8004          M                 bsf       ALUSTA,C             ; shift a 1 into d (result)
0216 1B1E          M nosub10         rlcf      ACCdLO, F
0217 1B1F          M                 rlcf      ACCdHI, F
  000B             M         variable i=i+1
                   M ;
0218 8804          M                 bcf       ALUSTA,C
0219 1B1A          M                 rlcf      ACCbLO, F
021A 1B1B          M                 rlcf      ACCbHI, F
021B 1B1C          M                 rlcf      ACCcLO, F
021C 1B1D          M                 rlcf      ACCcHI, F
021D 6A19          M                 movfp     ACCaHI,WREG
021E 041D          M                 subwf     ACCcHI,W             ; check if a>c
021F 9204          M                 btfss     ALUSTA,Z
0220 C223          M                 goto      notz11
0221 6A18          M                 movfp     ACCaLO,WREG
0222 041C          M                 subwf     ACCcLO,W             ; if msb equal then check lsb
0223 9004          M notz11          btfss     ALUSTA,C             ; carry set if c>a
0224 C22A          M                 goto      nosub11              ; if c < a
0225 6A18          M subca11         movfp     ACCaLO,WREG          ; c-a into c
0226 051C          M                 subwf     ACCcLO, F
0227 6A19          M                 movfp     ACCaHI,WREG
0228 031D          M                 subwfb    ACCcHI, F
0229 8004          M                 bsf       ALUSTA,C             ; shift a 1 into d (result)
022A 1B1E          M nosub11         rlcf      ACCdLO, F
022B 1B1F          M                 rlcf      ACCdHI, F
  000C             M         variable i=i+1
                   M ;
022C 8804          M                 bcf       ALUSTA,C
022D 1B1A          M                 rlcf      ACCbLO, F
022E 1B1B          M                 rlcf      ACCbHI, F
022F 1B1C          M                 rlcf      ACCcLO, F
0230 1B1D          M                 rlcf      ACCcHI, F
0231 6A19          M                 movfp     ACCaHI,WREG
0232 041D          M                 subwf     ACCcHI,W             ; check if a>c
0233 9204          M                 btfss     ALUSTA,Z
0234 C237          M                 goto      notz12
0235 6A18          M                 movfp     ACCaLO,WREG
0236 041C          M                 subwf     ACCcLO,W             ; if msb equal then check lsb
0237 9004          M notz12          btfss     ALUSTA,C             ; carry set if c>a
0238 C23E          M                 goto      nosub12              ; if c < a
```


© 1997 Microchip Technology Inc.

```
0239 6A18          M subca12          movfp     ACCaLO,WREG          ; c-a into c
023A 051C          M                  subwf     ACCcLO, F
023B 6A19          M                  movfp     ACCaHI,WREG
023C 031D          M                  subwfb    ACCcHI, F
023D 8004          M                  bsf       ALUSTA,C             ; shift a 1 into d (result)
023E 1B1E          M nosub12          rlcf      ACCdLO, F
023F 1B1F          M                  rlcf      ACCdHI, F
  000D             M         variable i=i+1
                   M ;
0240 8804          M                  bcf       ALUSTA,C
0241 1B1A          M                  rlcf      ACCbLO, F
0242 1B1B          M                  rlcf      ACCbHI, F
0243 1B1C          M                  rlcf      ACCcLO, F
0244 1B1D          M                  rlcf      ACCcHI, F
0245 6A19          M                  movfp     ACCaHI,WREG
0246 041D          M                  subwf     ACCcHI,W             ; check if a>c
0247 9204          M                  btfss     ALUSTA,Z
0248 C24B          M                  goto      notz13
0249 6A18          M                  movfp     ACCaLO,WREG
024A 041C          M                  subwf     ACCcLO,W             ; if msb equal then check lsb
024B 9004          M notz13           btfss     ALUSTA,C             ; carry set if c>a
024C C252          M                  goto      nosub13              ; if c < a
024D 6A18          M subca13          movfp     ACCaLO,WREG          ; c-a into c
024E 051C          M                  subwf     ACCcLO, F
024F 6A19          M                  movfp     ACCaHI,WREG
0250 031D          M                  subwfb    ACCcHI, F
0251 8004          M                  bsf       ALUSTA,C             ; shift a 1 into d (result)
0252 1B1E          M nosub13          rlcf      ACCdLO, F
0253 1B1F          M                  rlcf      ACCdHI, F
  000E             M         variable i=i+1
                   M ;
0254 8804          M                  bcf       ALUSTA,C
0255 1B1A          M                  rlcf      ACCbLO, F
0256 1B1B          M                  rlcf      ACCbHI, F
0257 1B1C          M                  rlcf      ACCcLO, F
0258 1B1D          M                  rlcf      ACCcHI, F
0259 6A19          M                  movfp     ACCaHI,WREG
025A 041D          M                  subwf     ACCcHI,W             ; check if a>c
025B 9204          M                  btfss     ALUSTA,Z
025C C25F          M                  goto      notz14
025D 6A18          M                  movfp     ACCaLO,WREG
025E 041C          M                  subwf     ACCcLO,W             ; if msb equal then check lsb
025F 9004          M notz14           btfss     ALUSTA,C             ; carry set if c>a
0260 C266          M                  goto      nosub14              ; if c < a
0261 6A18          M subca14          movfp     ACCaLO,WREG          ; c-a into c
0262 051C          M                  subwf     ACCcLO, F
0263 6A19          M                  movfp     ACCaHI,WREG
0264 031D          M                  subwfb    ACCcHI, F
0265 8004          M                  bsf       ALUSTA,C             ; shift a 1 into d (result)
0266 1B1E          M nosub14          rlcf      ACCdLO, F
0267 1B1F          M                  rlcf      ACCdHI, F
  000F             M         variable i=i+1
                   M ;
0268 8804          M                  bcf       ALUSTA,C
0269 1B1A          M                  rlcf      ACCbLO, F
026A 1B1B          M                  rlcf      ACCbHI, F
026B 1B1C          M                  rlcf      ACCcLO, F
026C 1B1D          M                  rlcf      ACCcHI, F
026D 6A19          M                  movfp     ACCaHI,WREG
026E 041D          M                  subwf     ACCcHI,W             ; check if a>c
026F 9204          M                  btfss     ALUSTA,Z
0270 C273          M                  goto      notz15
0271 6A18          M                  movfp     ACCaLO,WREG
0272 041C          M                  subwf     ACCcLO,W             ; if msb equal then check lsb
0273 9004          M notz15           btfss     ALUSTA,C             ; carry set if c>a
0274 C27A          M                  goto      nosub15              ; if c < a
```


© 1997 Microchip Technology Inc.

```
0275 6A18         M subca15         movfp     ACCaLO,WREG          ; c-a into c
0276 051C         M                 subwf     ACCcLO, F
0277 6A19         M                 movfp     ACCaHI,WREG
0278 031D         M                 subwfb    ACCcHI, F
0279 8004         M                 bsf       ALUSTA,C             ; shift a 1 into d (result)
027A 1B1E         M nosub15         rlcf      ACCdLO, F
027B 1B1F         M                 rlcf      ACCdHI, F
  0010            M       variable i=i+1
                  M      .endw
                  M ;
                00782 ;
                00783     #if SIGNED
                00784         btfss   sign,MSB                       ; negate (ACCc,ACCd)
                00785         return
                00786         negMac  ACCcLO,ACCcHI
                00787         negMac  ACCdLO,ACCdHI
                00788         return
                00789     #else
027C 0002       00790         return
                00791     #endif
                00792 ;
                00793         PAGE
                00794 ;*****************************************************************
                00795 ;
                00796 ;             Square Root By Newton Raphson Method
                00797 ;
                00798 ;    This routine computes the square root of a 16 bit number(with
                00799 ;  low byte in NumLo & high byte in NumHi ). After loading NumLo &
                00800 ;  NumHi with the desired number whose square root is to be computed,
                00801 ;  branch to location Sqrt ( by "GOTO  Sqrt" ). " CALL  Sqrt" cannot
                00802 ;  be issued because the Sqrt function makes calls to Math routines
                00803 ;  and the stack is completely used up.
                00804 ;       The result = sqrt(NumHi,NumLo) is returned in location SqrtLo.
                00805 ;  The total number of iterations is set to ten. If more iterations
                00806 ;  are desired, change "LupCnt equ .10" to the desired value. Also,
                00807 ;  the initial guess value of the square root is given set as
                00808 ;  input/2 ( in subroutine "init" ). The user may modify this scheme
                00809 ;  if a better initial approximation value is known. A good initial
                00810 ;  guess will help the algorithm converge at a faster rate and thus
                00811 ;  less number of iterations required.
                00812 ;       Two utility math routines are used by this program : D_divS
                00813 ;  and D_add. These two routines are listed as seperate routines
                00814 ;  under double precision Division and double precision addition
                00815 ;  respectively.
                00816 ;
                00817 ;  Note : If square root of an 8 bit number is desired, it is probably
                00818 ;         better to have a table look scheme rather than using numerical
                00819 ;         methods.
                00820 ;              This method is computationally quite intensive and
                00821 ;         slow, but very accurate and the convergence rate is high
                00822 ;
                00823 ;   Performance :
                00824 ;               Program Memory  :   22 (excluding D_divS subroutine)
                00825 ;               Clock Cycles    :   3000 (approximately,with 10 iterations)
                00826 ;
                00827 ;       The #of cycles depends on Number of Iterations Selected.
                00828 ; In a lot of cases 5 or less iterations may be sufficient
                00829 ;
                00830 ;
                00831 ;*****************************************************************
                00832 ;               Newton-Raphson Method
                00833 ;*************************************************************
                00834
027D            00835 Sqrt
027D E28B       00836         call    SqrtInit             ; compute initial sqrt = Num/2
027E            00837 nextIter
```

```
                00838    #if MODE_FAST
027E 7A24       00839          movfp    NumLo,ACCbLO
027F 7B25       00840          movfp    NumHi,ACCbHI
                00841    #else
                00842          movfp    NumLo,WREG
                00843          movwf    ACCbLO
                00844          movfp    NumHi,WREG
                00845          movwf    ACCbHI
                00846    #endif
                00847 ;
0280 E119       00848          call     D_divS          ; double precision division
                00849                                  ; double precision addition
0281 6A1E       00850          movfp    ACCdLO,WREG     ; ACCd + ACCa -> ACCd
0282 0F18       00851          addwf    ACCaLO, F       ; addwf lsb
0283 6A1F       00852          movfp    ACCdHI,WREG
0284 1119       00853          addwfc   ACCaHI, F       ; addwf msb
                00854                                  ; now divide by 2
0285 8804       00855          bcf      ALUSTA,C
0286 1919       00856          rrcf     ACCaHI, F
0287 1918       00857          rrcf     ACCaLO, F
                00858 ;
0288 1726       00859          decfsz   iterCnt, F
0289 C27E       00860          goto     nextIter
028A 0002       00861          return                   ; End Sqrt
                00862 ;
028B            00863 SqrtInit
028B B00A       00864          movlw    _LUPCNT
028C 0126       00865          movwf    iterCnt         ; set number of iterations
                00866   #if MODE_FAST
028D 7925       00867          movfp    NumHi,ACCaHI
028E 7824       00868          movfp    NumLo,ACCaLO
                00869   #else
                00870          movfp    NumHi,WREG
                00871          movwf    ACCaHI
                00872          movfp    NumLo,WREG       ; set initial guess root = NUM/2
                00873          movwf    ACCaLO
                00874   #endif
028F 8804       00875          bcf      ALUSTA,C
0290 1919       00876          rrcf     ACCaHI, F
0291 1918       00877          rrcf     ACCaLO, F        ; set initial sqrt = Num/2
0292 0002       00878          return
                00879 ;
                00880          PAGE
                00881 ;*****************************************************************
                00882 ;                    8x8 Software Multiplier
                00883 ;                ( Fast Version : Straight Line Code )
                00884 ;
                00885 ;   The 16 bit result is stored in 2 bytes
                00886 ;
                00887 ; Before calling the subroutine " mpy ", the multiplier should
                00888 ; be loaded in location " mulplr ", and the multiplicand in
                00889 ; " mulcnd " . The 16 bit result is stored in locations
                00890 ; H_byte & L_byte.
                00891 ;
                00892 ;       Performance :
                00893 ;                    Program Memory  :  36 words
                00894 ;                    # of cycles     :  36 (excluding call & return)
                00895 ;                    Scratch RAM     :   0 locations
                00896 ;                    W Register      :  Used
                00897 ;
                00898 ;  This routine is optimized for speed efficiency ( straight line code )
                00899 ;  For code efficiency, refer to "mult8x8S.asm" ( looped code )
                00900 ;*****************************************************************
                00901 ;     Define a macro for adding & right shifting
                00902 ;
                00903 multiply    MACRO
```


© 1997 Microchip Technology Inc.

```
                00904          variable  i ;
                00905            variable i = 0
                00906            .while i < 8
                00907                 btfsc     mulplr,i
                00908                 addwf    H_byte, F
                00909                 rrcf    H_byte, F
                00910                 rrcf    L_byte, F
                00911              variable i = i+1 ;
                00912            .endw
                00913          ENDM                    ; End of macro
                00914 ;
                00915 ;
0293            00916 mpy8x8_F
0293 291B       00917          clrf     H_byte, F
0294 291A       00918          clrf     L_byte, F
0295 6A18       00919          movfp    mulcnd,WREG     ; move the multiplicand to W reg.
0296 8804       00920          bcf      ALUSTA,C        ; Clear the carry bit in the status Reg.
                00921 ;
                00922          multiply
  0000             M           variable  i ;
  0000             M             variable i = 0
                   M             .while i < 8
0297 9819          M                  btfsc     mulplr,i
0298 0F1B          M                  addwf     H_byte, F
0299 191B          M                  rrcf      H_byte, F
029A 191A          M                  rrcf      L_byte, F
  0001             M               variable i = i+1 ;
029B 9919          M                  btfsc     mulplr,i
029C 0F1B          M                  addwf     H_byte, F
029D 191B          M                  rrcf      H_byte, F
029E 191A          M                  rrcf      L_byte, F
  0002             M               variable i = i+1 ;
029F 9A19          M                  btfsc     mulplr,i
02A0 0F1B          M                  addwf     H_byte, F
02A1 191B          M                  rrcf      H_byte, F
02A2 191A          M                  rrcf      L_byte, F
  0003             M               variable i = i+1 ;
02A3 9B19          M                  btfsc     mulplr,i
02A4 0F1B          M                  addwf     H_byte, F
02A5 191B          M                  rrcf      H_byte, F
02A6 191A          M                  rrcf      L_byte, F
  0004             M               variable i = i+1 ;
02A7 9C19          M                  btfsc     mulplr,i
02A8 0F1B          M                  addwf     H_byte, F
02A9 191B          M                  rrcf      H_byte, F
02AA 191A          M                  rrcf      L_byte, F
  0005             M               variable i = i+1 ;
02AB 9D19          M                  btfsc     mulplr,i
02AC 0F1B          M                  addwf     H_byte, F
02AD 191B          M                  rrcf      H_byte, F
02AE 191A          M                  rrcf      L_byte, F
  0006             M               variable i = i+1 ;
02AF 9E19          M                  btfsc     mulplr,i
02B0 0F1B          M                  addwf     H_byte, F
02B1 191B          M                  rrcf      H_byte, F
02B2 191A          M                  rrcf      L_byte, F
  0007             M               variable i = i+1 ;
02B3 9F19          M                  btfsc     mulplr,i
02B4 0F1B          M                  addwf     H_byte, F
02B5 191B          M                  rrcf      H_byte, F
02B6 191A          M                  rrcf      L_byte, F
  0008             M               variable i = i+1 ;
                   M             .endw
                00923 ;
02B7 0002       00924          return
                00925 ;
```


© 1997 Microchip Technology Inc.

```
                00926         PAGE
                00927 ;*****************************************************************
                00928 ;                    8x8 Software Multiplier
                00929 ;                ( Code Efficient : Looped Code )
                00930 ;
                00931 ;   The 16 bit result is stored in 2 bytes
                00932 ;
                00933 ; Before calling the subroutine " mpy ", the multiplier should
                00934 ; be loaded in location " mulplr ", and the multiplicand in
                00935 ; " mulcnd " . The 16 bit result is stored in locations
                00936 ; H_byte & L_byte.
                00937 ;
                00938 ;       Performance :
                00939 ;                     Program Memory  :  13 words (excluding call & return)
                00940 ;                     # of cycles     :  69       (excluding call & return)
                00941 ;                     Scratch RAM     :  1 byte
                00942 ;                     W Register      :  Used
                00943 ;
                00944 ;  This routine is optimized for code efficiency ( looped code )
                00945 ;  For time efficiency code refer to "mult8x8F.asm" ( straight line code )
                00946 ;*****************************************************************
                00947 ;
02B8            00948 mpy8x8_S
02B8 291B       00949         clrf    H_byte, F
02B9 291A       00950         clrf    L_byte, F
02BA 2922       00951         clrf    count, F
02BB 8322       00952         bsf     count,3         ; set count = 8
02BC 6A18       00953         movfp   mulcnd,WREG
02BD 8804       00954         bcf     ALUSTA,C        ; Clear the carry bit in the status Reg.
02BE            00955 loop
02BE 9819       00956         btfsc    mulplr,0
02BF 0F1B       00957         addwf    H_byte, F
02C0 191B       00958         rrcf     H_byte, F
02C1 191A       00959         rrcf     L_byte, F
02C2 2119       00960         rrncf    mulplr, F
02C3 1722       00961         decfsz   count, F
02C4 C2BE       00962         goto     loop
                00963 ;
02C5 0002       00964         return
                00965 ;
                00966         PAGE
                00967 ;*****************************************************************
                00968 ;                       Numerical Differenciation
                00969 ;
                00970 ;       The so called "Three-Point Formula" is implemented to
                00971 ; differenciate a sequence of points (uniformly sampled).
                00972 ;       The eqn implemented is :
                00973 ;             f'(Xn) = [ f(Xn - 2h) - 4*f(Xn - h) + 3*f(Xn)]*0.5/h
                00974 ;    where Xn is the present sample and 'h' is the step size.
                00975 ;
                00976 ;       The above formula may be rewritten as :
                00977 ;
                00978 ;         f'(Xn) = [ 0.5*f(Xn -2) - 2*f(Xn - 1) + 0.5*3*f(Xn)]*1/DiffK
                00979 ;  where DiffK = h = Step Size
                00980 ;
                00981 ;   This differenciation routine can be used very effectively
                00982 ; in the computation of the differential component part in
                00983 ; a PID Loop calculation in Motor Control Applications
                00984 ;
                00985 ;       Double precision arithmetic is used throught
                00986 ;  The present sample value is assumed to be in locations
                00987 ; (XnHi, XnLo). The past two values are assumed to be in locations
                00988 ; (Xn_1_Hi, Xn_1_Lo) & (Xn_2_Hi, Xn_2_Lo).
                00989 ;       The output value is located in DiffHi & DiffLo. No overflow
                00990 ; checking mechanism is implemented. If the values are limited
                00991 ; to 12 bits, then the user need not worry about overflows
```


© 1997 Microchip Technology Inc.

```
                00992 ;
                00993 ;  It is user's responsibility to update the past values with the
                00994 ; present values before calling this routine.
                00995 ;  After computation, the present value Xn is not moved to Xn_1
                00996 ; because the user may want these values to be intact for other
                00997 ; computations ( say numerical integration)
                00998 ;       Also it is user's responsibility to set past 2 values
                00999 ; (Xn_1 & Xn_2) values to be zero on initialization.
                01000 ;
                01001 ;*****************************************************************
                01002 ;
02C6            01003 Diff
02C6 6A2B       01004         movfp     Xn_2_Lo,WREG
02C7 0E27       01005         addwf     XnLo,W
02C8 011A       01006         movwf     ACCbLO
02C9 6A2C       01007         movfp     Xn_2_Hi,WREG
02CA 1028       01008         addwfc    XnHi,W
02CB 011B       01009         movwf     ACCbHI          ; Y = f(Xn-2) + f(Xn)
                01010 ;
02CC 6A27       01011         movfp     XnLo,WREG
02CD 0F1A       01012         addwf     ACCbLO, F
02CE 6A28       01013         movfp     XnHi,WREG
02CF 111B       01014         addwfc    ACCbHI, F
02D0 6A27       01015         movfp     XnLo,WREG
02D1 0F1A       01016         addwf     ACCbLO, F
02D2 6A28       01017         movfp     XnHi,WREG
02D3 111B       01018         addwfc    ACCbHI, F      ; Y = f(Xn-2) + 3*f(Xn)
                01019 ;
02D4 8804       01020         bcf       ALUSTA,C
02D5 191B       01021         rrcf      ACCbHI, F
02D6 191A       01022         rrcf      ACCbLO, F      ; Y = 0.5*[ f(Xn-2) + 3*f(Xn) ]
                01023
                01024 ;
02D7 6A29       01025         movfp     Xn_1_Lo,WREG
02D8 051A       01026         subwf     ACCbLO, F
02D9 6A2A       01027         movfp     Xn_1_Hi,WREG
02DA 031B       01028         subwfb    ACCbHI, F
02DB 6A29       01029         movfp     Xn_1_Lo,WREG
02DC 051A       01030         subwf     ACCbLO, F
02DD 6A2A       01031         movfp     Xn_1_Hi,WREG
02DE 031B       01032         subwfb    ACCbHI, F      ; Y = 0.5*[f(Xn-2) + 3*f(Xn)] - 2*f(Xn-1)
                01033 ;
02DF 6A2D       01034         movfp     DiffKLo,WREG
02E0 0118       01035         movwf     ACCaLO
02E1 6A2E       01036         movfp     DiffKHi,WREG
02E2 0119       01037         movwf     ACCaHI
                01038 ;
02E3 E119       01039         call      D_divS
02E4 6A1A       01040         movfp     ACCbLO,WREG
02E5 012F       01041         movwf     DiffLo
02E6 6A1B       01042         movfp     ACCbHI,WREG
02E7 0130       01043         movwf     DiffHi          ; result = Y/h
                01044 ;
02E8 0002       01045         return
                01046 ;
                01047         PAGE
                01048 ;*****************************************************************
                01049 ;
                01050 ;                         Numerical Integration
                01051 ;
                01052 ;
                01053 ;       Simpson's Three-Eighths Rule is implemented
                01054 ;
                01055 ;       Y(n) = [ f(X0) + 3*f(X1) + 3*f(X2) + f(X3)]*3*h/8
                01056 ;
                01057 ;  where 'h' is the step size and the integral is over the
```


© 1997 Microchip Technology Inc.

```
                01058 ;  range X0 to X3
                01059 ;       The above equation can be rewritten as
                01060 ;
                01061 ;       Y(n) = [ f(X0) + 3*f(X1) + 3*f(X2) + f(X3)]*IntgK
                01062 ;
                01063 ;   where IntgK = 3*h/8 (in locations (IntgKHi, IntgKHi)
                01064 ;
                01065 ;   This Integration routine can be used very effectively
                01066 ; in the computation of the integral component part in
                01067 ; a PID Loop calculation in Motor Control Applications
                01068 ;
                01069 ;       Double precision arithmetic is used throught
                01070 ;  The three input values over which the integral is to be computed
                01071 ; are assumed to be in locations (X0Lo,X0Hi), (X1Lo,X1Hi) , (X2Lo,X2Hi)
                01072 ; and (X3Lo,X3Hi)
                01073 ;       The output value is located in IntgHi & IntgLo. No overflow
                01074 ; checking mechanism is implemented. If the values are limited
                01075 ; to 12 bits, then the user need not worry about overflows
                01076 ;
                01077 ;  It is user's responsibility to update the past values with the
                01078 ; present values before calling this routine.
                01079 ;  After computation, the present value Xn is not moved to Xn_1
                01080 ; because the user may want these values to be intact for other
                01081 ; computations ( say numerical integration)
                01082 ;       Also it is user's responsibility to set past 2 values
                01083 ; (Xn_1 & Xn_2) values to be zero on initialization.
                01084 ;
                01085 ;
                01086 ;*********************************************************************
                01087 ;
02E9            01088 Integrate
02E9 6A31       01089         movfp     X0Lo,WREG
02EA 0E37       01090         addwf     X3Lo,W
02EB 011A       01091         movwf     ACCbLO
02EC 6A32       01092         movfp     X0Hi,WREG
02ED 1038       01093         addwfc    X3Hi,W
02EE 011B       01094         movwf     ACCbHI          ; Intg = f(X0) + f(X3)
                01095 ;
02EF 6A33       01096         movfp     X1Lo,WREG
02F0 0F1A       01097         addwf     ACCbLO, F
02F1 6A34       01098         movfp     X1Hi,WREG
02F2 111B       01099         addwfc    ACCbHI, F       ; Intg = f(X0) + f(X3) +X1
02F3 6A33       01100         movfp     X1Lo,WREG
02F4 0F1A       01101         addwf     ACCbLO, F
02F5 6A34       01102         movfp     X1Hi,WREG
02F6 111B       01103         addwfc    ACCbHI, F       ; Intg = f(X0) + f(X3) +2*X1
02F7 6A33       01104         movfp     X1Lo,WREG
02F8 0F1A       01105         addwf     ACCbLO, F
02F9 6A34       01106         movfp     X1Hi,WREG
02FA 111B       01107         addwfc    ACCbHI, F       ; Intg = f(X0) + f(X3) +3*X1
                01108 ;
02FB 6A35       01109         movfp     X2Lo,WREG
02FC 0F1A       01110         addwf     ACCbLO, F
02FD 6A36       01111         movfp     X2Hi,WREG
02FE 111B       01112         addwfc    ACCbHI, F       ; Intg = f(X0) + f(X3) +3*X1 + X2
02FF 6A35       01113         movfp     X2Lo,WREG
0300 0F1A       01114         addwf     ACCbLO, F
0301 6A36       01115         movfp     X2Hi,WREG
0302 111B       01116         addwfc    ACCbHI, F       ; Intg = f(X0) + f(X3) +3*X1 + 2*X2
0303 6A35       01117         movfp     X2Lo,WREG
0304 0F1A       01118         addwf     ACCbLO, F
0305 6A36       01119         movfp     X2Hi,WREG
0306 111B       01120         addwfc    ACCbHI, F       ; Intg = f(X0) + f(X3) +3*X1 + 3*X2
                01121 ;
0307 6A39       01122         movfp     IntgKLo,WREG
0308 0118       01123         movwf     ACCaLO
```


© 1997 Microchip Technology Inc.

```
0309 6A3A     01124         movfp     IntgKHi,WREG
030A 0119     01125         movwf     ACCaHI        ;  ACCa = IntgK (prepare for multiplication)
              01126 ;
030B E050     01127        call      D_mpyS       ; make sure to set for either SIGNED or UNSIGNED
030C 6A1E     01128         movfp     ACCdLO,WREG
030D 013B     01129         movwf     IntgLo        ; 32 bit result in ACCd & ACCc
030E 6A1F     01130         movfp     ACCdHI,WREG
030F 013C     01131         movwf     IntgHi        ; upper 16 bits = result
              01132 ;
0310 0002     01133         return
              01134 ;
              01135         PAGE
              01136 ;*****************************************************************
              01137 ;
              01138 ;                        Random Number Generator
              01139 ;
              01140 ; This routine generates a 16 Bit Pseudo Sequence Random Generator
              01141 ; It is based on Linear shift register feedback. The sequence
              01142 ; is generated by (Q15 xorwf Q14 xorwf Q12 xorwf Q3 )
              01143 ;
              01144 ;    The 16 bit random number is in location RandHi(high byte)
              01145 ;  & RandLo (low byte)
              01146 ;
              01147 ;       Before calling this routine, make sure the initial values
              01148 ; of RandHi & RandLo are NOT ZERO
              01149 ;       A good chiose of initial random number is 0x3045
              01150 ;*****************************************************************
              01151 ;
0311          01152 Random16
0311 1A19     01153         rlcf      RandHi,W
0312 0C19     01154         xorwf     RandHi,W
0313 1B0A     01155         rlcf      WREG, F       ; carry bit = xorwf(Q15,14)
              01156 ;
0314 1D19     01157         swapf     RandHi, F
0315 1C18     01158         swapf     RandLo,W
0316 230A     01159         rlncf     WREG, F
0317 0C19     01160         xorwf     RandHi,W      ; LSB = xorwf(Q12,Q3)
0318 1D19     01161         swapf     RandHi, F
0319 B501     01162         andlw     0x01
031A 1B18     01163         rlcf      RandLo, F
031B 0D18     01164         xorwf     RandLo, F
031C 1B19     01165         rlcf      RandHi, F
031D 0002     01166         return
              01167 ;
              01168         PAGE
              01169 ;*****************************************************************
              01170 ;                        Gaussian Noise Generator
              01171 ;
              01172 ; This routine generates a 16 Bit Gaussian distributed random
              01173 ; points. This routine calls the routine "Random16", which
              01174 ; generates a psuedo random noise sequence. Gaussian noise
              01175 ; is computed using the CENTRAL LIMIT THEOREM.
              01176 ;       The Central Limit Theorem states that the average weighted
              01177 ; sum of uncorelated samples tends to have a Gaussian distribution
              01178 ; For practical purposes, the sum could be over a sample size
              01179 ; of 32 Random numbers. Better results could result if a larger
              01180 ; sample size is desired. For faster results, a sum over 16 samples
              01181 ; would also be adequate ( say, for applications like Speech synthesis,
              01182 ; channel simulations, etc).
              01183 ;
              01184 ;       The 16 bit Gaussian distributed point is in locations
              01185 ; GaussHi & GaussLo
              01186 ;
              01187 ;       Before calling this routine, the initial seed of Random
              01188 ; number should be NON ZERO ( refer to notes on "Random16" routine
              01189 ;
```


© 1997 Microchip Technology Inc.

```
                01190 ;*******************************************************************
                01191 ;
031E            01192 Gauss
031E 2922       01193         clrf    count, F
031F 8522       01194         bsf     count,5         ; set Sample size = 32
0320 291A       01195         clrf    GaussLo, F
0321 291B       01196         clrf    GaussHi, F
0322 2920       01197         clrf    GaussTmp, F
                01198 ;
0323            01199 NextGauss
0323 E311       01200         call    Random16        ; get a random value
0324 6A18       01201         movfp   RandLo,WREG
0325 0F1A       01202         addwf   GaussLo, F
0326 6A19       01203         movfp   RandHi,WREG
0327 111B       01204         addwfc  GaussHi, F
0328 290A       01205         clrf    WREG, F
0329 1120       01206         addwfc  GaussTmp, F
032A 1722       01207         decfsz  count, F
032B C323       01208         goto    NextGauss       ; sum 16 random numbers
                01209 ;
032C B005       01210         movlw   5
032D            01211 GaussDiv16
032D 1920       01212         rrcf    GaussTmp, F
032E 191B       01213         rrcf    GaussHi, F
032F 191A       01214         rrcf    GaussLo, F          ; weghted average
0330 170A       01215         decfsz  WREG, F             ; divide by 32
0331 C32D       01216         goto    GaussDiv16
                01217 ;
0332 0002       01218         return
                01219 ;
                01220
                01221         END                     ; End Of arith.asm
0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0200 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0240 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0280 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
02C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0300 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXX-------------

All other memory blocks unused.

Program Memory Words Used:   819


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: BCD ARITHMETIC ROUTINES LISTING FILE OF BCD.ASM

```
MPASM 01.40 Released              BCD.ASM   1-16-1997  15:11:16         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001
                     00002     TITLE   "BCD Arithmetic Routines : Ver 1.0"
                     00003
                     00004 ;*****************************************************************
                     00005 ;                       BCD Arithmetic Routines
                     00006 ;
                     00007 ;       Program:          BCD.ASM
                     00008 ;       Revision Date:
                     00009 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00010 ;
                     00011 ;*****************************************************************
                     00012
                     00013         LIST     P = 17C42, columns=120, WRAP, L=0
                     00014
                     00015         include      "P17C42.INC"
                     00001         LIST
                     00002 ; P17C42.INC  Standard Header File, Version 1.03  Microchip Technology, Inc.
                     00264         LIST
                     00016
                     00017
                     00018         #define TRUE    1
                     00019         #define FALSE   0
                     00020
  00000001           00021 _INC    equ     1
  00000000           00022 _NO_INC equ     0
  00000000           00023 _LOW    equ     0
  00000001           00024 _HIGH   equ     1
                     00025
                     00026
                     00027         CBLOCK   0x20
  00000020           00028                 Lbyte, Hbyte
  00000022           00029                 R2, R1, R0       ; must maintain R2, R1, R0 sequence
  00000025           00030                 count
  00000026           00031                 Num1, Num2
                     00032         ENDC
                     00033 ;
  00000026           00034 BCD     equ      Num1
  00000026           00035 Htemp   equ      Num1
  00000027           00036 Ltemp   equ      Num2
                     00037 ;
                     00038         PAGE
                     00039
0000                 00040         ORG     0x0000
                     00041 ;*****************************************************************
                     00042 ;                       BCD Arithmetic Test Program
                     00043 ;*****************************************************************
                     00044 ;
0000                 00045 main
0000 2B21            00046         setf    Hbyte, F
0001 2B20            00047         setf    Lbyte, F
                     00048 ;                                ; 16 bit binary num = 0xffff
0002 E01F            00049         call    B2_BCD_Looped    ; after conversion the Decimal Num
                     00050 ;                                ; in R0, R1, R2 = 06,55,35
0003 2B21            00051         setf    Hbyte, F
```

```
0004 2B20      00052          setf     Lbyte, F
0005 E03F      00053          call     B2_BCD_Straight ; same as above, but straight line code
               00054 ;
0006 B006      00055          movlw    0x06
0007 0124      00056          movwf    R0
0008 B055      00057          movlw    0x55
0009 0123      00058          movwf     1
000A B035      00059          movlw    0x35
000B 0122      00060          movwf    R2              ; setf R0R1R2 = 65535
               00061 ;
000C E082      00062          call     BCDtoB          ; after conversion Hbyte = 0xff
               00063 ;                                 ; and Lbyte = 0xff
000D B099      00064          movlw    0x99
000E 0126      00065          movwf    Num1
000F B099      00066          movlw    0x99
0010 0127      00067          movwf    Num2            ; setf Num1 = Num2 = 0x99 (max BCD)
               00068 ;
0011 E093      00069          call     BCDAdd          ; after addition, Num2 = 98
               00070 ;                                 ; and Num1 = 01 ( 99+99 = 198)
               00071 ;
0012 B063      00072          movlw    0x63            ; setf Wreg = 63 hex
0013 E015      00073          call     BinBCD          ; after conversion, BCD = 99
               00074 ;                                 ; 63 hex = 99 decimal.
               00075 ;
0014 C014      00076 self     goto     self
               00077 ;
               00078          PAGE
               00079 ;*****************************************************************;
               00080 ;                  Binary To BCD Conversion Routine (8 bit)
               00081 ;
               00082 ;       This routine converts the 8 bit binary number in the W Reg
               00083 ; to a 2 digit BCD number in location BCD( compacted BCD Code)
               00084 ;       The least significant digit is returned in location LSD and
               00085 ; the most significant digit is returned in location MSD.
               00086 ;
               00087 ;   Performance :
               00088 ;               Program Memory  :  10
               00089 ;               Clock Cycles    :  62  (worst case when W = 63 Hex )
               00090 ;                                      ( i.e max Decimal number 99 )
               00091 ;*****************************************************************
               00092 ;
0015           00093 BinBCD
0015 2926      00094          clrf     BCD, F
0016           00095 again
0016 B1F0      00096          addlw    -10
0017 9004      00097          btfss    ALUSTA,C
0018 C01B      00098          goto     swapBCD
0019 1526      00099          incf     BCD, F
001A C016      00100          goto     again
001B           00101 swapBCD
001B B110      00102          addlw    10
001C 1D26      00103          swapf    BCD, F
001D 0926      00104          iorwf    BCD, F
001E 0002      00105          return
               00106 ;
               00107          PAGE
               00108 ;*****************************************************************
               00109 ;                  Binary To BCD Conversion Routine (16 Bit)
               00110 ;                        (LOOPED Version)
               00111 ;
               00112 ;      This routine converts a 16 Bit binary Number to a 5 Digit
               00113 ; BCD Number.
               00114 ;
               00115 ;       The 16 bit binary number is input in locations Hbyte and
               00116 ; Lbyte with the high byte in Hbyte.
               00117 ;       The 5 digit BCD number is returned in R0, R1 and R2 with R0
```


© 1997 Microchip Technology Inc.

```
                00118 ; containing the MSD in its right most nibble.
                00119 ;
                00120 ;   Performance :
                00121 ;               Program Memory  :  32
                00122 ;               Clock Cycles    :  750
                00123 ;
                00124 ;*****************************************************************;
                00125 ;
001F            00126 B2_BCD_Looped
001F 8404       00127         bsf     ALUSTA,FS0
0020 8504       00128         bsf     ALUSTA,FS1      ; set FSR0 for no auto increment
                00129 ;
0021 8804       00130         bcf     ALUSTA,C
0022 2925       00131         clrf    count, F
0023 8425       00132         bsf     count,4         ; set count = 16
0024 2924       00133         clrf    R0, F
0025 2923       00134         clrf    R1, F
0026 2922       00135         clrf    R2, F
0027            00136 loop16a
0027 1B20       00137         rlcf    Lbyte, F
0028 1B21       00138         rlcf    Hbyte, F
0029 1B22       00139         rlcf    R2, F
002A 1B23       00140         rlcf    R1, F
002B 1B24       00141         rlcf    R0, F
                00142 ;
002C 2725       00143         dcfsnz  count, F
002D 0002       00144         return
002E            00145 adjDEC
002E B022       00146         movlw   R2              ; load R2 as indirect address ptr
002F 0101       00147         movwf   FSR0
0030 E036       00148         call    adjBCD
                00149 ;
0031 1501       00150         incf    FSR0, F
0032 E036       00151         call    adjBCD
                00152 ;
0033 1501       00153         incf    FSR0, F
0034 E036       00154         call    adjBCD
                00155 ;
0035 C027       00156         goto    loop16a
                00157 ;
0036            00158 adjBCD
0036 6A00       00159         movfp   INDF0,WREG
0037 B103       00160         addlw   0x03
0038 9B0A       00161         btfsc   WREG,3          ; test if result > 7
0039 0100       00162         movwf   INDF0
003A 6A00       00163         movfp   INDF0,WREG
003B B130       00164         addlw   0x30
003C 9F0A       00165         btfsc   WREG,7          ; test if result > 7
003D 0100       00166         movwf   INDF0           ; save as MSD
003E 0002       00167         return
                00168 ;
                00169 ;*****************************************************************
                00170 ;               Binary To BCD Conversion Routine (16 Bit)
                00171 ;                     (Partial Straight Line Version)
                00172 ;
                00173 ;      This routine converts a 16 Bit binary Number to a 5 Digit
                00174 ; BCD Number.
                00175 ;
                00176 ;       The 16 bit binary number is input in locations Hbyte and
                00177 ; Lbyte with the high byte in Hbyte.
                00178 ;       The 5 digit BCD number is returned in R0, R1 and R2 with R0
                00179 ; containing the MSD in its right most nibble.
                00180 ;
                00181 ;   Performance :
                00182 ;               Program Memory  :  44
                00183 ;               Clock Cycles    :  572
```


© 1997 Microchip Technology Inc.

```
                00184 ;
                00185 ;*******************************************************************;
                00186 ;
003F            00187 B2_BCD_Straight
003F 8404       00188         bsf     ALUSTA,FS0
0040 8504       00189         bsf     ALUSTA,FS1      ; set FSR0 for no auto increment
                00190 ;
0041 8804       00191         bcf     ALUSTA,C
0042 2925       00192         clrf    count, F
0043 8425       00193         bsf     count,4         ; set count = 16
0044 2924       00194         clrf    R0, F
0045 2923       00195         clrf    R1, F
0046 2922       00196         clrf    R2, F
0047            00197 loop16b
0047 1B20       00198         rlcf    Lbyte, F
0048 1B21       00199         rlcf    Hbyte, F
0049 1B22       00200         rlcf    R2, F
004A 1B23       00201         rlcf    R1, F
004B 1B24       00202         rlcf    R0, F
                00203 ;
004C 2725       00204         dcfsnz  count, F
004D 0002       00205         return                  ; DONE
004E B022       00206         movlw   R2              ; load R2 as indirect address ptr
004F 0101       00207         movwf   FSR0
                00208 ; adjustBCD
0050 6A00       00209         movfp   INDF0,WREG
0051 B103       00210         addlw   0x03
0052 9B0A       00211         btfsc   WREG,3          ; test if result > 7
0053 0100       00212         movwf   INDF0
0054 6A00       00213         movfp   INDF0,WREG
0055 B130       00214         addlw   0x30
0056 9F0A       00215         btfsc   WREG,7          ; test if result > 7
0057 0100       00216         movwf   INDF0           ; save as MSD
                00217 ;
0058 1501       00218         incf    FSR0, F
                00219 ; adjustBCD
0059 6A00       00220         movfp   INDF0,WREG
005A B103       00221         addlw   0x03
005B 9B0A       00222         btfsc   WREG,3          ; test if result > 7
005C 0100       00223         movwf   INDF0
005D 6A00       00224         movfp   INDF0,WREG
005E B130       00225         addlw   0x30
005F 9F0A       00226         btfsc   WREG,7          ; test if result > 7
0060 0100       00227         movwf   INDF0           ; save as MSD
                00228 ;
0061 1501       00229         incf    FSR0, F
                00230 ; adjustBCD
0062 6A00       00231         movfp   INDF0,WREG
0063 B103       00232         addlw   0x03
0064 9B0A       00233         btfsc   WREG,3          ; test if result > 7
0065 0100       00234         movwf   INDF0
0066 6A00       00235         movfp   INDF0,WREG
0067 B130       00236         addlw   0x30
0068 9F0A       00237         btfsc   WREG,7          ; test if result > 7
0069 0100       00238         movwf   INDF0           ; save as MSD
                00239 ;
006A C047       00240         goto    loop16b
                00241 ;
                00242         PAGE
                00243 ;*************************************************************************
                00244 ;               BCD To Binary Conversion
                00245 ;
                00246 ;       This routine converts a 5 digit BCD number to a 16 bit binary
                00247 ; number.
                00248 ;       The input 5 digit BCD numbers are asumed to be in locations
                00249 ; R0, R1 & R2 with R0 containing the MSD in its right most nibble.
```


© 1997 Microchip Technology Inc.

```
                00250 ;
                00251 ;       The 16 bit binary number is output in registers Hbyte & Lbyte
                00252 ; ( high byte & low byte repectively ).
                00253 ;
                00254 ;       The method used for conversion is :
                00255 ;               input number X = abcde ( the 5 digit BCD number )
                00256 ;       X = (R0,R1,R2) = abcde = 10[10[10[10a+b]+c]+d]+e
                00257 ;
                00258 ;   Performance :
                00259 ;               Program Memory  :  30
                00260 ;               Clock Cycles    :  112
                00261 ;
                00262 ;*****************************************************************;
                00263 ;
006B            00264 mpy10b
006B B50F       00265         andlw     0x0f
006C 0F20       00266         addwf     Lbyte, F
006D 9804       00267         btfsc     ALUSTA,C
006E 1521       00268         incf      Hbyte, F
006F            00269 mpy10a
006F 8804       00270         bcf       ALUSTA,C          ; multiply by 2
0070 1A20       00271         rlcf      Lbyte,W
0071 0127       00272         movwf     Ltemp
0072 1A21       00273         rlcf      Hbyte,W           ; (Htemp,Ltemp) = 2*N
0073 0126       00274         movwf     Htemp
                00275 ;
0074 8804       00276         bcf       ALUSTA,C          ; multiply by 2
0075 1B20       00277         rlcf      Lbyte, F
0076 1B21       00278         rlcf      Hbyte, F
0077 8804       00279         bcf       ALUSTA,C          ; multiply by 2
0078 1B20       00280         rlcf      Lbyte, F
0079 1B21       00281         rlcf      Hbyte, F
007A 8804       00282         bcf       ALUSTA,C          ; multiply by 2
007B 1B20       00283         rlcf      Lbyte, F
007C 1B21       00284         rlcf      Hbyte, F          ; (Hbyte,Lbyte) = 8*N
                00285 ;
007D 6A27       00286         movfp     Ltemp,WREG
007E 0F20       00287         addwf     Lbyte, F
007F 6A26       00288         movfp     Htemp,WREG
0080 1121       00289         addwfc    Hbyte, F
0081 0002       00290         return                      ; (Hbyte,Lbyte) = 10*N
                00291 ;
                00292 ;
0082            00293 BCDtoB
0082 2921       00294         clrf      Hbyte, F
0083 6A24       00295         movfp     R0,WREG
0084 B50F       00296         andlw     0x0f
0085 0120       00297         movwf     Lbyte
0086 E06F       00298         call      mpy10a          ; result = 10a+b
                00299 ;
0087 1C23       00300         swapf     R1,W
0088 E06B       00301         call      mpy10b          ; result = 10[10a+b]
                00302 ;
0089 6A23       00303         movfp     R1,WREG
008A E06B       00304         call      mpy10b          ; result = 10[10[10a+b]+c]
                00305 ;
008B 1C22       00306         swapf     R2,W
008C E06B       00307         call      mpy10b          ; result = 10[10[10[10a+b]+c]+d]
                00308 ;
008D 6A22       00309         movfp     R2,WREG
008E B50F       00310         andlw     0x0f
008F 0F20       00311         addwf     Lbyte, F
0090 9804       00312         btfsc     ALUSTA,C
0091 1521       00313         incf      Hbyte, F        ; result = 10[10[10[10a+b]+c]+d]+e
0092 0002       00314         return                    ; BCD to binary conversion done
                00315 ;
```

```
                00316         PAGE
                00317 ;*******************************************************************;
                00318 ;
                00319 ;                 Unsigned BCD Addition
                00320 ;
                00321 ;       This routine performs a 2 Digit Unsigned BCD Addition
                00322 ; It is assumed that the two BCD numbers to be added are in
                00323 ; locations Num1 & Num2. The result is the sum of Num1+Num2
                00324 ; and is stored in location Num2 and the overflow carry is returned
                00325 ; in location Num1
                00326 ;
                00327 ;   Performance :
                00328 ;               Program Memory  :       5
                00329 ;               Clock Cycles    :       5
                00330 ;
                00331 ;*******************************************************************;
                00332 ;
0093            00333 BCDAdd
0093 6A26       00334         movfp    Num1,WREG
0094 0E27       00335         addwf     Num2,W          ; perform binary addition
0095 2F27       00336         daw     Num2, F           ; adjust for BCD addition
0096 2926       00337         clrf     Num1, F
0097 1B26       00338         rlcf     Num1, F          ; set Num1 = carry bit
0098 0002       00339         return
                00340 ;
                00341 ;*******************************************************************
                00342 ;
                00343         END
BCD Arithmetic Routines : Ver 1.0
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXX------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   153


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C: FXP_DIV.ASM

```
MPASM 01.40 Released            FLOAT.ASM   1-16-1997  15:11:46         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

             00001
             00002         TITLE   "Binary Floating Arithmetic Routines For PIC17C42 : Ver 1.0"
             00003
             00004         LIST    P = 17C42, columns=120, WRAP, L=0
             00005 ;
             00006         include       "p17c42.inc"
             00001         LIST
             00002 ; P17C42.INC Standard Header File, Version 1.03  Microchip Technology, Inc.
             00264         LIST
             00007
             00008         #define TRUE    1
             00009         #define FALSE   0
             00010
             00011         #define LSB     0
             00012         #define MSB     7
             00013 ;
             00014 ;*****************************************************************
             00015 ;               Binary Floating Point Addition, Subtraction And
             00016 ;                       Multiplication Routines
             00017 ;
             00018 ;  Mantissa = 16 bits
             00019 ;  Exponent = 8 bits     ( exponent is binary and not decimal)
             00020 ;                         i.e a number ABCD EXP(X) = 0xABCD * (2**X)
             00021 ;
             00022 ;       Before calling any of the following floating point routines,
             00023 ;  it is required to set Indirect Register 0 ( FSR0 ) for
             00024 ;  No-Autoincrement ( i.e. Set bits FS0 & FS1 in ALUSTA to 1s)
             00025 ;
             00026 ;
             00027 ;       Program:          FLOAT.ASM
             00028 ;       Revision Date:
             00029 ;                         1-13-97   Compatibility with MPASMWIN 1.40
             00030 ;
             00031 ;*****************************************************************;
             00032 ;
             00033         CBLOCK 0x20
  00000020   00034                ACCaLO, ACCaHI, EXPa
  00000023   00035                ACCbLO, ACCbHI, EXPb
  00000026   00036                ACCcLO, ACCcHI
  00000028   00037                ACCdLO, ACCdHI
  0000002A   00038                temp,   sign
             00039         ENDC
             00040 ;
  00000001   00041 Mode16  equ     TRUE     ; Change this to FALSE for 32 bit product
             00042
             00043         PAGE
             00044 ;
0000         00045         ORG     0x0000
             00046 ;
             00047 ;*****************************************************************
             00048 ;                 Floating Point Routines Test Program
             00049 ;*****************************************************************
             00050 ;
0000         00051 main
```

```
                00052 ;
0000 8404       00053           bsf     ALUSTA,FS0      ; set FSR0 for no autoincrement
0001 8504       00054           bsf     ALUSTA,FS1
                00055 ;
0002 E009       00056           call    loadAB          ; result of adding
ACCb(EXPb)+ACCa(EXPa)->ACCb(EXPb)
0003 E019       00057           call    F_add           ; Here Accb = 403F, EXPb = 07
                00058 ;
0004 E009       00059           call    loadAB          ; result of subtracting
ACCb(EXPb)-ACCa(EXPa)->ACCb(EXPb)
0005 E016       00060           call    F_sub           ; Here Accb = 7F7F, EXPb = 06
                00061 ;
0006 E009       00062           call    loadAB          ; result of multiplying ACCb(EXPb) *
ACCa(EXPa)->ACCb(EXPb)
0007 E03B       00063           call    F_mpy           ; Here ACCb = FF7E, EXPb = 12
                00064 ;
0008 C008       00065 self      goto    self
                00066 ;
                00067 ;     Load constant values to (ACCa, EXPa) & (ACCb, EXPb) for testing
                00068 ;
0009            00069 loadAB
0009 B001       00070           movlw   0x01
000A 0121       00071           movwf   ACCaHI
000B B0FF       00072           movlw   0xff            ; loads ACCa = 01FF EXP(4)
000C 0120       00073           movwf   ACCaLO
000D B004       00074           movlw   0x04
000E 0122       00075           movwf   EXPa
                00076 ;
000F B07F       00077           movlw   0x7f
0010 0124       00078           movwf   ACCbHI
0011 B0FF       00079           movlw   0xff            ; loads ACCb = 7fff EXP(6)
0012 0123       00080           movwf   ACCbLO
0013 B006       00081           movlw   0x06
0014 0125       00082           movwf   EXPb
0015 0002       00083           return
                00084 ;
                00085           PAGE
                00086 ;*****************************************************************
                00087 ;       Floating Point Subtraction  ( ACCb - ACCa -> ACCb )
                00088 ;
                00089 ;   Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
                00090 ;       (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits ) with
                00091 ;           the 8 bit exponent in EXPa .
                00092 ;       (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits ) with
                00093 ;           the 8 bit exponent in EXPb .
                00094 ;       (c) CALL F_sub
                00095 ;       (d) The result is in location ACCbLO & ACCbHI ( 16 bits ) with
                00096 ;           the 8 bit exponent in EXPb.
                00097 ;
                00098 ;*****************************************************************;
                00099 ;
0016            00100 F_sub
0016 B020       00101           movlw   ACCaLO
0017 0101       00102           movwf     FSR0
0018 E075       00103           call    negate          ; At first negate ACCa; Then addwf
                00104 ;
                00105 ;*****************************************************************
                00106 ;       Floating Point Addition  ( ACCb + ACCa -> ACCb )
                00107 ;
                00108 ;   Addition :  ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
                00109 ;       (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits ) with
                00110 ;           the 8 bit exponent in EXPa.
                00111 ;       (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits ) with
                00112 ;           the 8 bit exponent in EXPb.
                00113 ;       (c) CALL F_add
                00114 ;       (d) The result is in location ACCbLO & ACCbHI ( 16 bits ) with
```


© 1997 Microchip Technology Inc.

```
                        00115 ;          the 8 bit exponent in EXPb
                        00116 ;
                        00117 ;*****************************************************************
                        00118 ;
0019                    00119 F_add
0019 6A22               00120         movfp     EXPa,WREG
001A 3125               00121         cpfseq    EXPb
001B C01D               00122         goto      Lbl1
001C C026               00123         goto      noAddNorm          ; if exponents are equal
001D                    00124 Lbl1
001D 3025               00125         cpfslt    EXPb
001E E09E               00126         call      F_swap             ; if B > A then swap ( A<->B )
001F 6A22               00127         movfp     EXPa,WREG
0020 0525               00128         subwf     EXPb, F
0021                    00129 scloop
0021 E030               00130         call      addNorm
0022 1F25               00131         incfsz    EXPb, F
0023 C021               00132         goto      scloop
0024 6A22               00133         movfp     EXPa,WREG
0025 0125               00134         movwf     EXPb
0026                    00135 noAddNorm
0026 6A21               00136         movfp     ACCaHI,WREG
0027 0824               00137         iorwf     ACCbHI,W
0028 012B               00138         movwf     sign               ; save the sign ( MSB states)
0029 E036               00139         call      D_add              ; compute double precision integer addwf
002A 972B               00140         btfss     sign,MSB
002B 9724               00141         btfss     ACCbHI,MSB
002C 0002               00142         return
002D 1525               00143         incf      EXPb, F
002E 8804               00144         bcf       ALUSTA,C
002F C033               00145         goto      shftR
                        00146 ;
0030                    00147 addNorm
0030 8804               00148         bcf       ALUSTA,C
0031 9F24               00149         btfsc     ACCbHI,MSB
0032 8004               00150         bsf       ALUSTA,C           ; set carry if < 0
0033                    00151 shftR
0033 1924               00152         rrcf      ACCbHI, F
0034 1923               00153         rrcf      ACCbLO, F
0035 0002               00154         return
                        00155 ;
                        00156         PAGE
                        00157 ;*****************************************************************
                        00158 ;       Double Precision Addition
                        00159 ;
0036                    00160 D_add
0036 6A20               00161         movfp     ACCaLO,WREG
0037 0F23               00162         addwf     ACCbLO, F          ; addwf lsb
0038 6A21               00163         movfp     ACCaHI,WREG
0039 1124               00164         addwfc    ACCbHI, F          ; addwf msb with carry
003A 0002               00165         return
                        00166 ;
                        00167         PAGE
                        00168 ;*****************************************************************
                        00169 ;               Binary Floating Point Multiplication
                        00170 ;
                        00171 ;   Multiplication :
                        00172 ;       ACCb(16 bits)EXP(b) * ACCa(16 bits)EXPa -> ACCb(16 bits)EXPb
                        00173 ;   where, EXP(x) represents an 8 bit exponent.
                        00174 ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits ) with
                        00175 ;                                an 8 bit exponent in location EXPa
                        00176 ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits ) with
                        00177 ;                                an 8 bit exponent in location EXPb
                        00178 ;      (c) CALL F_mpy
                        00179 ;      (d) The 16 bit result overwrites ACCb(ACCbLO & ACCbHI). The exponent
                        00180 ;              is stored in EXPb and the results are normalized.
```


© 1997 Microchip Technology Inc.

```
                00181 ;
                00182 ; NOTE : If one needs to get a 32 bit product( & an 8 bit exponent ),
                00183 ;        re assemble the program after changing the line " Mode16 equ TRUE"
                00184 ;        to " Mode16  equ  FALSE ".
                00185 ;        If this option is chosen, then the 32 bit result is returned in
                00186 ;        ( ACCbHI, ACCbLO, ACCcHI, ACCcLO ) and the 8 bit exponent in EXPb.
                00187 ;        This method ( with " Mode16 equ FALSE " ) is NOT Recommended.
                00188 ;
                00189 ;  If a 32 bit mantissa is desired, set MODE16 equ FALSE
                00190 ;*****************************************************************
                00191 ;
003B            00192 F_mpy
003B E07D       00193         call    S_SIGN
003C E064       00194         call    setup
003D            00195 mloop
003D 8804       00196         bcf     ALUSTA,C
003E 1929       00197         rrcf    ACCdHI, F           ; rotate d right
003F 1928       00198         rrcf    ACCdLO, F
0040 9804       00199         btfsc   ALUSTA,C
0041 E036       00200         call    D_add
0042 1924       00201         rrcf    ACCbHI, f
0043 1923       00202         rrcf    ACCbLO, F
0044 1927       00203         rrcf    ACCcHI, F
0045 1926       00204         rrcf    ACCcLO, F
0046 172A       00205         decfsz  temp, F             ; loop until all bits checked
0047 C03D       00206         goto    mloop
                00207 ;
0048 6A22       00208         movfp   EXPa,WREG
0049 0F25       00209         addwf   EXPb, F
                00210 ;
                00211     #if Mode16
004A 3324       00212         tstfsz  ACCbHI
004B C05B       00213         goto    finup               ; if ACCbHI != 0
004C 3323       00214         tstfsz  ACCbLO
004D C055       00215         goto    Shft08              ; if ACCbLO != 0 && ACCbHI == 0
                00216 ;
004E 6A27       00217         movfp   ACCcHI,WREG
004F 0124       00218         movwf   ACCbHI              ; if ACCb == 0, then move ACCc to ACCb
0050 6A26       00219         movfp   ACCcLO,WREG
0051 0123       00220         movwf   ACCbLO
0052 B016       00221         movlw   16
0053 0F25       00222         addwf   EXPb, F
0054 C05B       00223         goto    finup
                00224 ;
0055            00225 Shft08
0055 6A23       00226         movfp   ACCbLO,WREG
0056 0124       00227         movwf   ACCbHI
0057 6A27       00228         movfp   ACCcHI,WREG
0058 0123       00229         movwf   ACCbLO
0059 B008       00230         movlw   8
005A 0F25       00231         addwf   EXPb, F
                00232 ;
                00233     #endif                          ; matching endif for IF Mode16
                00234 ;
005B            00235 finup
005B 972B       00236         btfss   sign,MSB
005C C08B       00237         goto    F_norm
                00238 ;
005D B026       00239         movlw   ACCcLO
005E 0101       00240         movwf   FSR0
005F E075       00241         call    negate
0060 B023       00242         movlw   ACCbLO
0061 0101       00243         movwf   FSR0
0062 E075       00244         call    negate
0063 C08B       00245         goto    F_norm              ; normalize floating point
                00246 ;
```


© 1997 Microchip Technology Inc.

```
                00247 ;*******************************************************************
                00248 ;
0064            00249 setup
0064 292A       00250         clrf    temp, F
0065 842A       00251         bsf     temp,4              ; set temp = 16
0066 6A24       00252         movfp   ACCbHI,WREG         ; move ACCb to ACCd
0067 0129       00253         movwf   ACCdHI
0068 6A23       00254         movfp   ACCbLO,WREG
0069 0128       00255         movwf   ACCdLO
006A 2924       00256         clrf    ACCbHI, F
006B 2923       00257         clrf    ACCbLO, F           ; clear ACCb ( ACCbLO & ACCbHI )
006C 0002       00258         return
                00259 ;
                00260         PAGE
                00261 ;*******************************************************************
                00262 ;       Double Precision Negate Routines
                00263 ;
006D            00264 negateAlt
006D 6A00       00265         movfp   INDF0,WREG
006E 8D04       00266         bcf     ALUSTA,FS1
006F 2D00       00267         negw    INDF0, F
0070 8504       00268         bsf     ALUSTA,FS1
0071 6A00       00269         movfp   INDF0,WREG
0072 2900       00270         clrf    INDF0, F
0073 0300       00271         subwfb  INDF0, F
0074 0002       00272         return
                00273 ;
0075            00274 negate
0075 1300       00275         comf    INDF0, F
0076 8D04       00276         bcf     ALUSTA,FS1
0077 1500       00277         incf    INDF0, F
0078 8504       00278         bsf     ALUSTA,FS1
0079 9A04       00279         btfsc   ALUSTA,Z
007A 0700       00280         decf    INDF0, F
007B 1300       00281         comf    INDF0, F
007C 0002       00282         return
                00283 ;
                00284         PAGE
                00285 ;*******************************************************************
                00286 ;   Check Sign of the Number, if so negate and set the SIGN flag
                00287 ;
007D            00288 S_SIGN
007D 6A21       00289         movfp   ACCaHI,WREG
007E 0C24       00290         xorwf   ACCbHI,W
007F 012B       00291         movwf   sign
0080 9724       00292         btfss   ACCbHI,MSB          ; if MSB set go & negate ACCb
0081 C085       00293         goto    chek_A
                00294 ;
0082 B023       00295         movlw   ACCbLO
0083 0101       00296         movwf   FSR0
0084 E075       00297         call    negate
                00298 ;
0085            00299 chek_A
0085 9721       00300         btfss   ACCaHI,MSB          ; if MSB set go & negate ACCa
0086 0002       00301         return
0087 B020       00302         movlw   ACCaLO
0088 0101       00303         movwf   FSR0
0089 E075       00304         call    negate
008A 0002       00305         return
                00306 ;
                00307         PAGE
                00308 ;*******************************************************************
                00309 ;      Normalize Routine
                00310 ; Normalizes ACCb for use in floating point calculations.
                00311 ; Call this routine as often as possible to minimize the loss
                00312 ; of precission. This routine normalizes ACCb so that the
```


© 1997 Microchip Technology Inc.

```
                00313 ; mantissa is maximized and the exponent minimized.
                00314 ;
                00315 ;*****************************************************************
                00316 ;
008B            00317 F_norm                                ; normalize ACCb
008B 3324       00318         tstfsz   ACCbHI
008C C090       00319         goto     C_norm
008D 3323       00320         tstfsz   ACCbLO
008E C090       00321         goto     C_norm
008F 0002       00322         return
                00323 ;
0090            00324 C_norm
0090 9E24       00325         btfsc    ACCbHI,MSB-1
0091 0002       00326         return
0092 E095       00327         call     shftSL
0093 0725       00328         decf     EXPb, F
0094 C090       00329         goto     C_norm
                00330 ;
0095            00331 shftSL
0095 8804       00332         bcf      ALUSTA,C
                00333 ;
                00334     #if   Mode16
0096 1B26       00335         rlcf     ACCcLO, F
0097 1B27       00336         rlcf     ACCcHI, F
                00337     #endif
                00338 ;
0098 1B23       00339         rlcf     ACCbLO, F
0099 1B24       00340         rlcf     ACCbHI, F
009A 8F24       00341         bcf      ACCbHI,MSB
009B 9804       00342         btfsc    ALUSTA,C
009C 8724       00343         bsf      ACCbHI,MSB
009D 0002       00344         return
                00345 ;
                00346 ;*****************************************************************
                00347 ;  Swap ACCa & ACCb   [ (ACCa,EXPa) <--> (ACCb,EXPb) ]
                00348 ;
009E            00349 F_swap
009E 6A21       00350         movfp    ACCaHI,WREG
009F 012A       00351         movwf    temp
00A0 6A24       00352         movfp    ACCbHI,WREG        ; ACCaHI <--> ACCbHI
00A1 0121       00353         movwf    ACCaHI
00A2 6A2A       00354         movfp    temp,WREG
00A3 0124       00355         movwf    ACCbHI
                00356 ;
00A4 6A20       00357         movfp    ACCaLO,WREG
00A5 012A       00358         movwf    temp
00A6 6A23       00359         movfp    ACCbLO,WREG        ; ACCaLO <--> ACCbLO
00A7 0120       00360         movwf    ACCaLO
00A8 6A2A       00361         movfp    temp,WREG
00A9 0123       00362         movwf    ACCbLO
                00363 ;
00AA 6A25       00364         movfp    EXPb,WREG
00AB 012A       00365         movwf    temp
00AC 6A22       00366         movfp    EXPa,WREG          ; EXPa <--> EXPb
00AD 0125       00367         movwf    EXPb
00AE 6A2A       00368         movfp    temp,WREG
00AF 0122       00369         movwf    EXPa
00B0 0002       00370         return
                00371 ;
                00372 ;*****************************************************************
                00373 ;               Normalizes A Floating Point Number
                00374 ;       The number is assumed to be  (LowByte, HighByte, Exp) in
                00375 ; consecutive locations. Before calling this routine, the address
                00376 ; of the LowByte should be loaded into FSR0 (indirect register ptr)
                00377 ;*****************************************************************
                00378 ;
```


© 1997 Microchip Technology Inc.

```
00B1            00379 Normalize
00B1 1501       00380         incf     FSR0, F
00B2 3300       00381         tstfsz   INDF0
00B3 C0B9       00382         goto     NextNorm1
00B4 0701       00383         decf     FSR0, F
00B5 3300       00384         tstfsz   INDF0
00B6 C0B8       00385         goto     NextNorm2
00B7 0002       00386         return
00B8            00387 NextNorm2
00B8 1501       00388         incf     FSR0, F
00B9            00389 NextNorm1
00B9 9E00       00390         btfsc    INDF0,MSB-1
00BA 0002       00391         return
00BB E0C0       00392         call     shiftNorm
00BC 1501       00393         incf     FSR0, F
00BD 0700       00394         decf     INDF0, F
00BE 0701       00395         decf     FSR0, F
00BF C0B9       00396         goto     NextNorm1
                00397 ;
00C0            00398 shiftNorm
00C0 8804       00399         bcf      ALUSTA,C
00C1 1501       00400         incf     FSR0, F
00C2 1B00       00401         rlcf     INDF0, F
00C3 1501       00402         incf     FSR0, F
00C4 1B00       00403         rlcf     INDF0, F
00C5 0701       00404         decf     FSR0, F
00C6 0701       00405         decf     FSR0, F
00C7 0701       00406         decf     FSR0, F
00C8 1B00       00407         rlcf     INDF0, F
00C9 1501       00408         incf     FSR0, F
00CA 1B00       00409         rlcf     INDF0, F
00CB 8F00       00410         bcf      INDF0,MSB
00CC 9804       00411         btfsc    ALUSTA,C
00CD 8700       00412         bsf      INDF0,MSB
00CE 0002       00413         return
                00414 ;
                00415
                00416         END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXX- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   207


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Using the Capture Module

Author: *Mark Palmer*
Microchip Technology Inc.

INTRODUCTION

The PICmicro™ family of RISC microcontrollers has been designed to provide advanced performance and a cost-effective solution for a variety of applications. This application note provides examples which illustrate the uses of input capture using the PIC17C42 Timer3 module. These examples may be modified to suit the specific needs of an application.

This application note has 4 examples that use the Timer3 input capture. They are:

- Frequency Counter (Period Measurement)
- Frequency Counter (Period Measurement) using a Free Running Timer
- Pulse Width Measurement
- Frequency Counter (Period Measurement) with Input Prescaler

TIMER3 DESCRIPTION

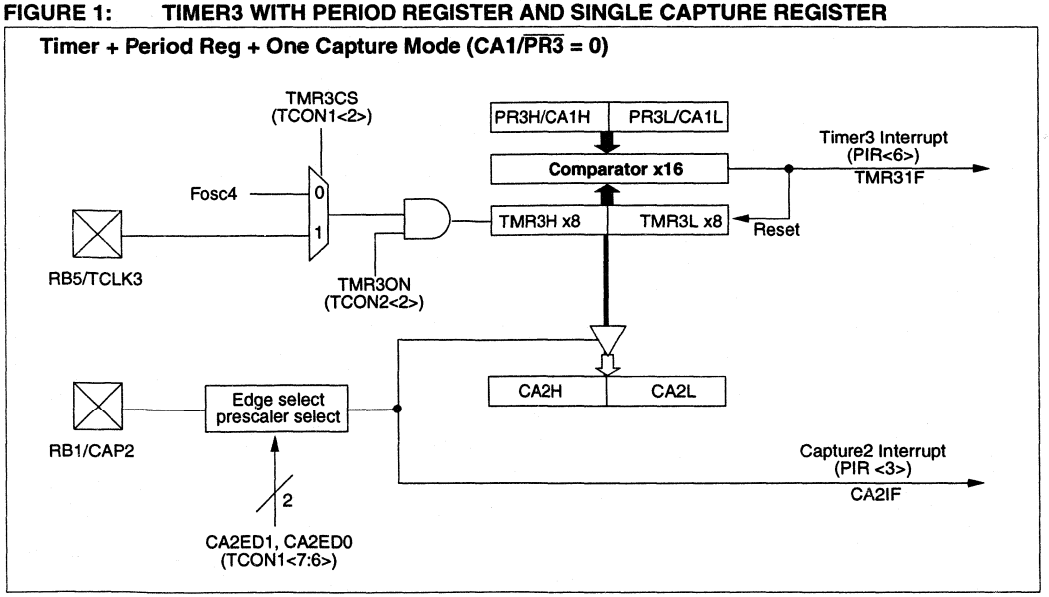
Timer3 is a 16-bit timer/counter that has two modes of operation which are software selected. The CA1/$\overline{PR3}$ bit (TCON2<3>) selects the mode of operation. The two modes are:

- Timer3 with Period Register and Single Capture Register (Figure 1).
- Timer3 and Dual Capture Registers (Figure 2).

Timer3 is the time-base for capture operations.

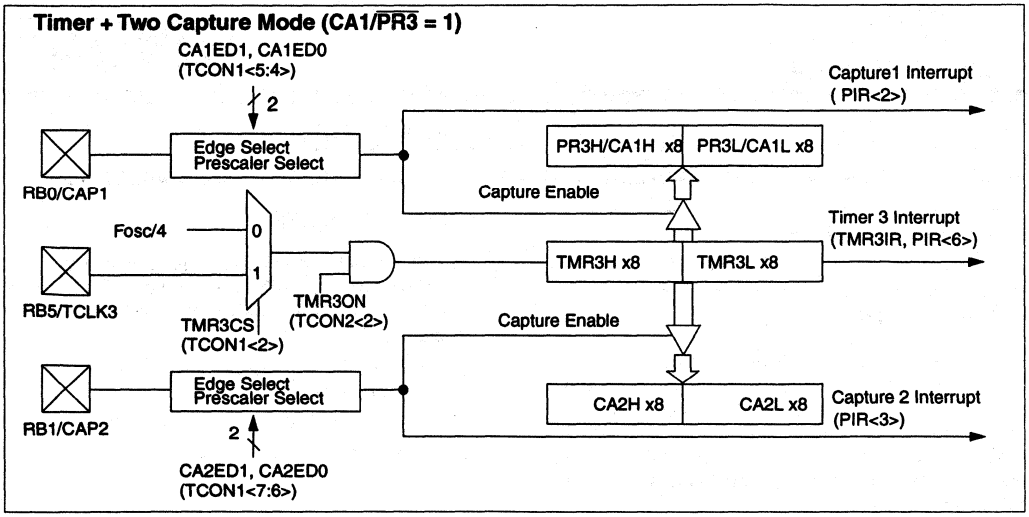
4

FIGURE 1: TIMER3 WITH PERIOD REGISTER AND SINGLE CAPTURE REGISTER



© 1997 Microchip Technology Inc.

FIGURE 2: TIMER3 AND DUAL CAPTURE REGISTERS



The period register allows the time base of Timer3 to be something other than the 2^{16} counter overflow value, which corresponds to FFFFh (65536) cycles. This is accomplished by loading the desired period value into the PR3H/CA1H:PR3L/CA1L register pair. The overflow time can be calculated by this equation:

TOFL = TCLK • (value in PR3H/CA1H:PR3L/CA1L register pair + 1).

Where TCLK is either the internal system clock (TCY) or the external clock cycle time. Table 1 the shows time-out periods for different period values at different frequencies. The values in the register are the closest approximation for the period value. All examples in this application note uses a Timer3 overflow value of FFFFh.

© 1997 Microchip Technology Inc.

TABLE 1: TIMER3 OVERFLOW TIMES

	Period Register					
Overflow Time	**@ 16 MHz (250 ns)**	**@ 10 MHz (400 ns)**	**@ 8 MHz (500 ns)**	**@5 MHz (500 ns)**	**@2 MHz (2.0 µs)**	**@32 kHz (125 µs)**
8.192 s	N.A.	N.A.	N.A.	N.A.	N.A.	0xFFFF
131.072 ms	N.A.	N.A.	N.A.	N.A.	0xFFFF	0x0418
52.428 ms	N.A.	N.A.	N.A.	0xFFFF	0x6666	0x01A3
32.7675 ms	N.A.	N.A.	0xFFFF	0x9FFF	0x3FFF	0x0106
26.214 ms	N.A.	0xFFFF	0xCE20	0x80D4	0x3388	0x00D3
16.384 ms	0xFFFF	0xA000	0x8000	0x5000	0x2000	0x0083
10.0 ms	0x9C40	0x61A8	0x4E20	0x30D4	0x1388	0x0050
4.0 ms	0x3E80	0x2710	0x1F40	0x1388	0x07D0	0x0020
1.0 ms	0x0FA0	0x09C4	0x07D0	0x04E2	0x01F4	0x0008
600 µs	0x0960	0x05DC	0x04B0	0x02EE	0x012C	0x0005
100 µs	0x0190	0x00FA	0x00C8	0x007D	0x0032	N.A.

The uses of an input capture are all for time based measurements. These include:

- Frequency measurement
- Duty cycle and pulse width measurements

The PIC17C42 has two pins (RB0/CAP1 and RB1/CAP2) which can be used for capturing the Timer3 value, when a specified edge occurs. The input capture can be specified to occur on one of the following four events:

- Falling Edge
- Rising Edge
- 4th Rising Edge
- 16th Rising Edge

These are specified bits 7:6 for CAP2 and 5:4 for CAP2 by the register TCON1<7:4>.

This flexibility allows an interface without the need of additional hardware to change polarity or specify an input prescaler.

© 1997 Microchip Technology Inc.

The control registers that are used for by Timer3 are shown in Table 2. Shaded Boxes are control bits that are not used by the Timer3 module, the Peripheral Interrupt enable and flag bits, and the Global Interrupt enable bit.

TABLE 2: REGISTERS ASSOCIATED WITH TIMER3 AND CAPTURE

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other resets (Note1)
16h, Bank 3	TCON1	CA2ED1	CA2ED0	CA1ED1	CA1ED0	T16	TMR3CS	TMR2CS	TMR1CS	0000 0000	0000 0000
17h, Bank 3	TCON2	CA2OVF	CA1OVF	PWM2ON	PWM1ON	CA1/PR3	TMR3ON	TMR2ON	TMR1ON	0000 0000	0000 0000
10h, Bank 2	TMR1	Timer1 register								xxxx xxxx	uuuu uuuu
11h, Bank 2	TMR2	Timer2 register								xxxx xxxx	uuuu uuuu
16h, Bank 1	PIR	RBIF	TMR3IF	TMR2IF	TMR1IF	CA2IF	CA1IF	TXIF	RCIF	0000 0010	0000 0010
17h, Bank 1	PIE	RBIE	TMR3IE	TMR2IE	TMR1IE	CA2IE	CA1IE	TXIE	RCIE	0000 0000	0000 0000
07h, Unbanked	INTSTA	PEIF	T0CKIF	T0IF	INTF	PEIE	T0CKIE	T0IE	INTE	0000 0000	0000 0000
06h, Unbanked	CPUSTA	—	—	STKAV	GLINTD	TO	PD	—	—	--11 11--	--11 qq--
14h, Bank 2	PR1	Timer1 period register								xxxx xxxx	uuuu uuuu
15h, Bank 2	PR2	Timer2 period register								xxxx xxxx	uuuu uuuu
10h, Bank 3	PW1DCL	DC1	DC0	—	—	—	—	—	—	xx-- ----	uu-- ----
11h, Bank 3	PW2DCL	DC1	DC0	TM2PW2	—	—	—	—	—	xx0- ----	uu0- ----
12h, Bank 3	PW1DCH	DC9	DC8	DC7	DC6	DC5	DC4	DC3	DC2	xxxx xxxx	uuuu uuuu
13h, Bank 3	PW2DCH	DC9	DC8	DC7	DC6	DC5	DC4	DC3	DC2	xxxx xxxx	uuuu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented read as a '0', q - value depends on condition, shaded cells are not used by Timer1 or Timer2.

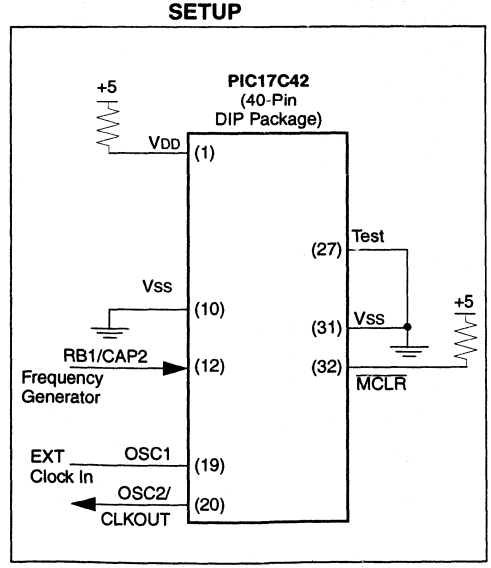
Note 1: Other (non power-up) resets include: external reset through $\overline{MCLR}$ and WDT Timer Reset.

This Application Note has 4 examples that USE the Timer3 input capture. They are:

- Frequency Counter (Period Measurement)
- Frequency Counter (Period Measurement) using a Free Running Timer
- Pulse Width Measurement
- Frequency Counter (Period Measurement) with Input Prescaler

All these examples can be run from a simple setup, this is shown in Figure 3.

FIGURE 3: APPLICATION HARDWARE SETUP



A discussion of each application with the operation of the software and application issues. The source listings for these are in appendices A-D.

© 1997 Microchip Technology Inc.

PERIOD MEASUREMENT (FREQUENCY COUNTER)

Period measurement is simply done by clearing the counter to 0000h, then starting the counter on the 1st rising edge. On the following rising edge, the capture2 register is loaded with the Timer3 value and the TMR3 register CA2H:CA2L is cleared. If the period is greater than the overflow rate of TMR3, the register overflows, causing an interrupt. With a TMR3 overflow, an interrupt occurs and the overflow counter may need to be incremented. The overflow counter should be incremented if:

- The TMR3 overflow is the only interrupt source.
- Both the TMR3 overflow and capture2 interrupts occurred at near the same time, but the TMR3 overflow occurred first, Most Significant Byte of the Capture2 register is cleared (CA24-00h)).

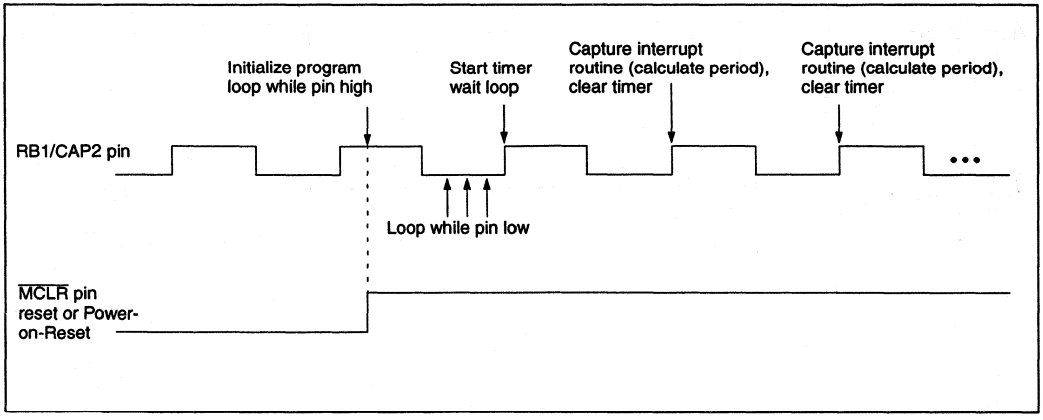
Once a capture has occurred, the capture registers are moved to data RAM, the capture2 interrupt flag is cleared and the TMR3 register is loaded with an offset value. This offset value is the number of cycles from the time the interrupt routine is entered to when the TMR3 register is reloaded. In this example a data RAM location is used as an overflow counter. This gives in effect a 24-bit timer. The software flow for this routine is shown in Figure 4.

The program listing in Appendix A implements this, assuming only TMR3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

Code size:	30 Words
RAM used:	4 Bytes
Maximum frequency that can be measured:	130 kHz
Minimum frequency that can be measured:	0.25 Hz
Measurement Accuracy:	± TCY (± 250 ns)

4

FIGURE 4: SOFTWARE TIMING FLOW RELATIVE TO INPUT SIGNAL ON RB1/CAP2 PIN



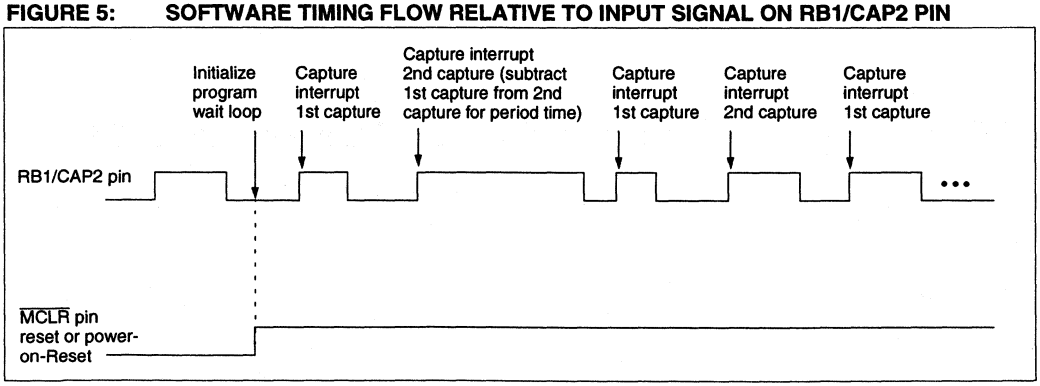
PERIOD MEASUREMENT (FREQUENCY COUNTER) USING A FREE RUNNING TIMER

In many applications the timer would need to be used for multiple tasks, and is required not to be reset (modified) by any one of these tasks. This is called a free running timer. To do period measurement in an application with a free-running timer, the program needs to store each capture in a data RAM location pair (word). The 1st capture in data RAM locations input capture2A (IC2AH:IC2AL) and the 2nd capture in data RAM locations input capture2B (IC2BH:IC2BL). Once the two captures have occurred, the values in these two words are subtracted. Since this is a free running timer, the value in input capture2B may be less than the value in input capture2A. This is if the 1st capture occurs, then the TMR3 overflows, and then the 2nd capture occurs. So an overflow counter should only be incremented if the TMR3 overflow occurs after a capture1 but before the capture2 occurs. With the use of an overflow counter this becomes an effective 24-bit period counter. The software flow for this routine is shown in Figure 5.

The program listing in Appendix B implements this, assuming only TMR3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

Code size:	41 Words
RAM used:	7 Bytes
Maximum frequency that can be measured:	71 kHz
Minimum frequency that can be measured:	0.25 Hz
Measurement Accuracy:	± TCY (± 250 ns)

FIGURE 5: SOFTWARE TIMING FLOW RELATIVE TO INPUT SIGNAL ON RB1/CAP2 PIN



© 1997 Microchip Technology Inc.

PULSE WIDTH MEASUREMENT USING A FREE RUNNING TIMER

Applications that require the measurement of a pulse width can also be easily handled. The PIC17C42 can be programmed to measure either the low or the high pulse time. The software example in Appendix C measures the high pulse time. The program is initialized to capture on the rising edge of the RB1/CAP2 pin. After this event occurs, the capture mode is switched to the falling edge of the RB1/CAP2 pin. When the capture edge is modified (rising to falling, or falling to rising) a capture interrupt is generated. This "false" interrupt request must be cleared before leaving the interrupt service routine, or the program will immediately re-enter the interrupt service routine due to this "false" request. When the falling edge of the RB1/CAP2 pin occurs, the difference of the two capture values is calculated. The flow for this is shown in Figure 6.

Due to the software overhead of the peripheral interrupt routine the following are the limitations on the input signal on the RB1/CAP2 pin. This does not include any software overhead that may be required in the main routine, or if additional peripheral interrupt features need to be included. This is shown in Table 3. If you assume that the input is a square wave (high time = low time), one needs to take the worst case time of the two minimum pulse times (11 µs) times two, to determine the period. The maximum continuous input frequency would then be approximately 45.5 kHz. For a single pulse measurement, minimum pulse width is 4.5 µs.

The program listing in Appendix C implements this, assuming only TMR3 overflow and capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

Code size:	51 Words
RAM used:	7 Bytes
Maximum frequency that can be measured:	71 kHz
Minimum frequency that can be measured:	0.25 Hz
Measurement Accuracy:	± TCY (± 250 ns)

4

FIGURE 6: SOFTWARE TIMING FLOW RELATIVE TO INPUT SIGNAL ON RB1/CAP2

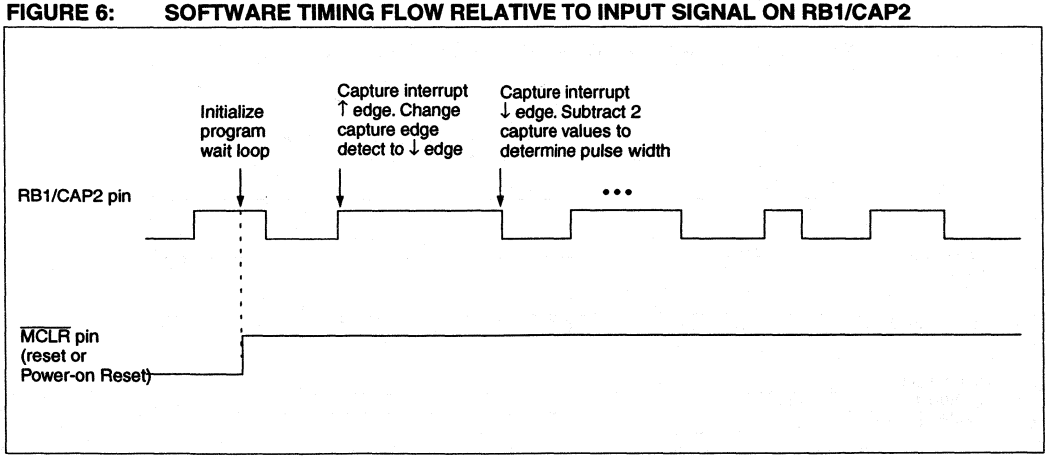


TABLE 3: PERIPHERAL INTERRUPT ROUTINE

EVENT		# of Cycles	Time @ 16 MHz
1st CAPTURE	Capture1 only Capture1 and Timer Overflow	18 30	4.5 µs 7.5 µs
2nd CAPTURE	Capture only Capture and Timer Overflow	35 41	8.75 µs 10.25 µs
Minimum Pulse High	Capture1 and Timer Overflow + INT Latency	33	8.25 µs
Minimum Pulse Low	Capture2 and Timer Overflow + INT Latency	44	11 µs
Minimum Period (square wave)	2 • (Minimum Pulse Low)	88	22 µs

© 1997 Microchip Technology Inc.

PERIOD MEASUREMENT (FREE RUNNING TIMER) WITH A PRESCALER

Occasionally the application may require a prescaler on the input signal. This may be due to application requirements, such as:

- Require higher resolution measurement of the input signal
- Reduce interrupt service overhead
- The input frequency is higher than interrupt service routine

The software selectable prescaler of the PIC17C42 allows the designer to easily implement this in their system without the cost of additional hardware. Care must be taken in determining if this option is appropriate. For example, if the input frequency is not stable (excessive frequency change per period) then the prescaler will give a less accurate capture value than the individual measurements.

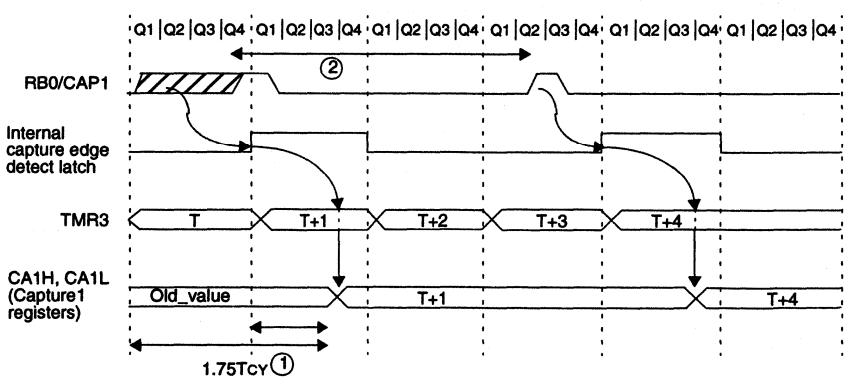
In cases where the resolution of the input frequency is important, the prescaler can be used to reduce the input capture error. There are two components to input capture:

- Resolution Error
- Input Synchronization Error

These two errors combine to form the total capture error. Resolution error is dependent on the rate at which the timer is incremented. Remember the timer may be based on an external clock (must be slower than TCY). The input synchronization error is dependent on the system clock speed (TCY), and will be less than TCY.

It is easy to see that when a capture occurs the synchronization error (TESYSC) can be up to 1 TCY (Figure 7). This error is constant regardless of the number of edges that occur before the capture is taken. So a capture on the 1st edge gives a synchronization error per sample up to TCY. While a capture taken on the 16th edge gives a synchronization error per sample only up to TCY / 16, by achieving a smaller percentage of error, the captured value becomes more accurate.

FIGURE 7: SYNCHRONIZATION ERROR WITH NO CAPTURE PRESCALER



Note 1: Capture edge to actual register update latency is 1.75 TCY maximum, 0.75 TCY minimum. This implies that when measuring a pulse or a period, the measurement error is ± TCY.

Note 2: With no prescaler on the input capture, two consecutive capturing edges must be apart by at least TCY. This allows the internal "capture edge detect latch" to reset.

© 1997 Microchip Technology Inc.

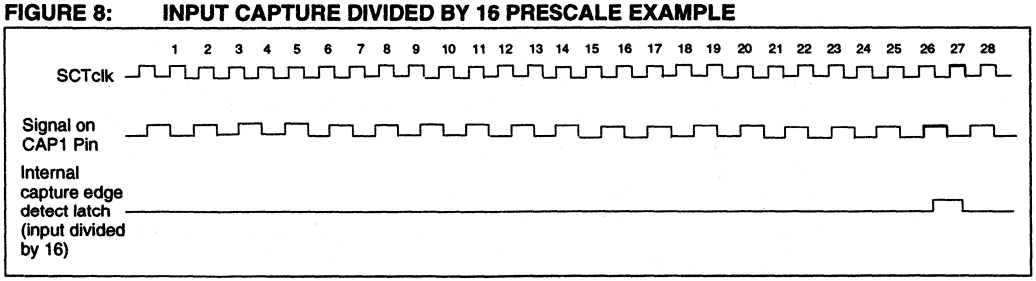
Another scenario is when the signal on the input capture pin is different than the system cycle time (TCY), for example 1.6TCY. If you tried to capture this you would have a capture value of 1. If you set the prescaler to actually capture on the 16th edge you would have 16 * 1.6 TCY = 25.6TCY, which would be latched on the 26th TCY (Figure 8). This 0.4 TCY error is over 16 samples, which therefore gives an effective error/sample of 0.025TCY.

The program listing in Appendix D implements this, assuming only TMR3 overflow and Capture2 interrupt sources. This example may be modified to suit the particular needs of your application. The following is a performance summary for this program (@ 16 MHz):

Code size:	41 Words
RAM used:	7 Bytes
Maximum frequency that can be measured:	80 kHz
Minimum frequency that can be measured:	0.25 Hz
Measurement Accuracy:	± TCY (± 16.625 ns)

4

FIGURE 8: INPUT CAPTURE DIVIDED BY 16 PRESCALE EXAMPLE



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: PERIOD MEASUREMENT EXAMPLE CODE

```
MPASM 01.40 Released          IC_D16_2.ASM   1-16-1997  15:15:17          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 17C42, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;
                     00005 ; This is the basic outline for a program that can determine the
                     00006 ; frequency of an input, via input capture. The input capture has
                     00007 ; been selected to capture on the 16th rising edge. This is useful
                     00008 ; for high frequency inputs, where an interrupt on each rising edge
                     00009 ; would not be able to be serviced (at that rate). This particular
                     00010 ; example can support an input signal with a period of approximatly
                     00011 ; 625 nS. Without the divide by 16 selected, this is approximatly
                     00012 ; 10 us. This period time increases (frequency decreases) as the
                     00013 ; overhead in the main routine increases.
                     00014 ;
                     00015 ; This routine uses an 8-bit register to count the times that timer3
                     00016 ; overflowed. At the Max crystal frequency of 16 MHz, this gives an
                     00017 ; overflow time of (16)(2**8 + 1)(2**16)(250 nS) > 67.37 sec. If
                     00018 ; measurement of longer time intervals is required, the overflow
                     00019 ; counter could be extended to 16 (or more) bits.
                     00020 ;
                     00021 ; Timer 3 in this example is a free running timer. The input
                     00022 ; capture is generated on the RB1/CAP2 pin. There is a flag
                     00023 ; that specifies if this is the 1st or 2nd capture.
                     00024 ; The first capture is the start of the period measurement. The
                     00025 ; second capture value gives the end of the period. In this type
                     00026 ; of measurement If the 2nd capture value < the 1st captue value
                     00027 ; then the overflow counter should be decremented.
                     00028 ;
                     00029 ;       Program:          IC_D16_2.ASM
                     00030 ;       Revision Date:
                     00031 ;                         1-14-97    Compatibility with MPASMWIN 1.40
                     00032 ;
                     00033 ;*****************************************************************
                     00034 ;
                     00035 ;
                     00036 ; Do the EQUate table
                     00037 ;
00000020             00038 IC2OF           EQU     0x20        ; T3 overflow register
00000021             00039 IC2BH           EQU     0x21        ; T3 ICA2 MSB register (2nd Cap)
00000022             00040 IC2BL           EQU     0x22        ; T3 ICA2 LSB register
00000023             00041 IC2AH           EQU     0x23        ; T3 ICB2 MSB register (1st Cap)
00000024             00042 IC2AL           EQU     0x24        ; T3 ICB2 LSB register
00000025             00043 T3OFLCNTR       EQU     0x25        ; Temperay T3 overflow register
                     00044 ;
00000026             00045 FLAG_REG        EQU     0x26        ; Register that has the Flag bits
                     00046 ;
                     00047 ;    FLAG_REG  bit   7   6   5   4   3   2   1   0
                     00048 ;                    -   -   -   -   -   -  UFL CAP1
                     00049 ;    CAP1 = 0,  1st Capture
                     00050 ;         = 1,  2nd Capture
                     00051 ;
                     00052 ;    UFL = 0,  No Underflow
                     00053 ;        = 1,  Underflow during subtract
                     00054 ;
000007FF             00055 END_OF_PROG_MEM         EQU 0x07FF
```


© 1997 Microchip Technology Inc.

```
                    00056 ;
                    00057 ;
  00000004          00058 ALUSTA          EQU     0x04
  00000006          00059 CPUSTA          EQU     0x06
  00000007          00060 INTSTA          EQU     0x07
  0000000A          00061 W               EQU     0x0A
                    00062 ;
  00000012          00063 PORTB           EQU     0x12        ; Bank 0
                    00064 ;
  00000016          00065 PIR             EQU     0x16        ; Bank 1
  00000017          00066 PIE             EQU     0x17
                    00067 ;
  00000012          00068 TMR3L           EQU     0x12        ; Bank 2
  00000013          00069 TMR3H           EQU     0x13
  00000016          00070 T3PRL           EQU 0x16
  00000017          00071 T3PRH           EQU 0x17
                    00072 ;
  00000014          00073 CA2L            EQU     0x14        ; Bank 3
  00000015          00074 CA2H            EQU     0x15
  00000016          00075 TCON1           EQU     0x16
  00000017          00076 TCON2           EQU     0x17
                    00077  PAGE
0000                00078                 ORG     0x0000      ; Origin for the RESET vector
0000 C028           00079                 GOTO    START       ; On reset, go to the start of
                    00080                                     ;   the program
0008                00081                 ORG     0x0008      ; Origin for the external RA0/INT
                    00082                                     ;   interrupt vector
0008 C068           00083                 GOTO    EXT_INT     ; Goto the ext. interrupt
                    00084                                     ;   on RA0/INT routine
0010                00085                 ORG    0x0010       ; Origin for the TMR0
                    00086                                     ;   overflow interrupt vector
0010 C069           00087                 GOTO    TMR0INT     ; Goto the TMR0 overflow interrupt
                    00088                                     ;   routine
0018                00089                 ORG     0x0018      ; Origin for the external
                    00090                                     ;   RB1/T0CKI interrupt vector
0018 C06A           00091                 GOTO    T0INT      ; Goto the ext. interrupt on
                    00092                                     ;   RB1/T0CKI routine
0020                00093                 ORG     0x0020      ; Origin for the interrupt vector
                    00094                                     ;   of any enabled peripheral
0020 C03E           00095                 GOTO    PER_INT     ; Goto the interrupt from a
                    00096                                     ;   peripheral routine
                    00097  PAGE
0028                00098                 ORG     0x0028      ; Origin for the top of
                    00099                                     ;   program memory
0028 8406           00100 START           BSF     CPUSTA,4    ; Disable ALL interrupts via the
                    00101                                     ;   Global Interrupt Disable
                    00102                                     ;   (GLINTD) bit.
                    00103                                     ;
0029                00104 MAIN                                ; Place Main program here
0029 B803           00105                 MOVLB   3           ; Select register Bank 3
002A 2817           00106                 CLRF    TCON2,0     ; Stop the timers, Single Capture
002B B0F0           00107                 MOVLW   0x0F0       ; Initalize TCON1 so that
002C 0116           00108                 MOVWF   TCON1       ; T1 (8-bit), T2 (8-bit),
                    00109                                     ; and T3 run off the internal
                    00110                                     ; system clock. Capture2 captures
                    00111                                     ; on the 16th rising edge.
                    00112 ;
                    00113 ; Initialize Timer 3, load the timer with the number of cycles that
                    00114 ; the device executes (from RESET) before the timer is turned on
                    00115 ; Therefore the offset is required due to software overhead.
                    00116 ;
002D B802           00117                 MOVLB   2           ; Select register Bank 2
002E 280A           00118                 CLRF    W,0         ; Clear the W register
002F 0126           00119                 MOVWF   FLAG_REG    ; Initalize to 0
0030 0113           00120                 MOVWF   TMR3H       ; Timer3 MSB = 0
0031 B000           00121                 MOVLW   0x00        ; Timer3 LSB = Offset
```

```
0032 0112          00122                MOVWF   TMR3L       ;
                   00123 ;
                   00124 ; Load the Timer 3 period register with 0xFFFF, which will give an
                   00125 ; interrupt on the overflow of Timer3
                   00126 ;
0033 B0FF          00127                MOVLW   0xFF        ;
0034 0117          00128                MOVWF   T3PRH       ;
0035 0116          00129                MOVWF   T3PRL       ;
                   00130 ;
                   00131 ; the timer should be started and interrupts enabled.
                   00132 ;
0036 B803          00133                MOVLB   3           ; Select register Bank 3
0037 8217          00134                BSF     TCON2,2     ; Turn on timer 3.
0038 8307          00135                BSF     INTSTA,3    ; Turn on Peripheral Interrupts
0039 B801          00136                MOVLB   1           ; Select register Bank 1
003A B048          00137                MOVLW   0x48        ; Enable Caputure 2 and Timer3
003B 0117          00138                MOVWF   PIE         ;    Interrupts (when GLINTD = 0)
                   00139 ;
                   00140 ; This is where you would do the things you wanted to do.
                   00141 ; this example will only loop waiting for the interrupts.
                   00142 ;
003C 8C06          00143 WAIT           BCF     CPUSTA,4    ; Enable ALL interrupts
003D C03C          00144                GOTO    WAIT        ; Loop here waiting for a timer
                   00145                                    ;   Interrupt
                   00146  PAGE
                   00147 ;
                   00148 ; The interrupt routine for any peripheral interrupt, This routine
                   00149 ; only deals with Timer3 (T3) interrupts.
                   00150 ;
                   00151 ; Time required to execute interrupt routine. Not including
                   00152 ; interrupt latency (time to enter into the interrupt routine)
                   00153 ;
                   00154 ;       case1 - only T3 overflow             = 12 cycles
                   00155 ;       case2 - 1st capture                  = 14 cycles
                   00156 ;       case3 - 2nd capture                  = 30 cycles
                   00157 ;       case4 - T3 overflow and 1st capture  = 34 cycles
                   00158 ;       case5 - T3 overflow and 2nd capture  = 50 cycles
                   00159 ;
                   00160 ;
003E B801          00161 PER_INT        MOVLB   1           ; Select register Bank 1
003F 9E16          00162                BTFSC   PIR,6       ; Did T3 overflow?
                   00163                                    ; If not skip next Instruction
0040 C055          00164                GOTO    T3OVFL      ; Inc overflow cntr and clear flag
0041 9316          00165 CK_CAP         BTFSS   PIR,3       ; Did the RB1/CAP2 pin cause an
                   00166                                    ;   interrupt?
0042 0005          00167                RETFIE              ; No RB1/CAP2 interrupt,
                   00168                                    ;   Return from Interrupt
                   00169 ;
                   00170 ; This potion of the code takes the 1st capture and stores its
                   00171 ; value in register pair IC2AH:IC2AL. When the 2nd capture
                   00172 ; is take, its value is stored in register pair IC2BH:IC2BL.
                   00173 ; A 16-bit subtract is performed, with the final 24-bit result
                   00174 ; being stored in IC2OF:IC2BH:IC2BL. This value will no longer
                   00175 ; be correct after the next capture occurs (IC2BH:IC2BL will
                   00176 ; change), so the main routine must utilize this value before
                   00177 ; it changes.
                   00178 ;
0043 8B16          00179 CAPTURE        BCF     PIR,3       ; Clear Capture2 interrupt flag
0044 B803          00180                MOVLB   3           ; Select register Bank 3
0045 9826          00181                BTFSC   FLAG_REG,0  ; 1st or 2nd capture2?
0046 C04B          00182                GOTO    CAP2        ; It was the 2nd Capture
0047 5424          00183 CAP1           MOVPF   CA2L,IC2AL  ; Move the captured value to
0048 5523          00184                MOVPF   CA2H,IC2AH  ;   temporary registers
0049 8026          00185                BSF     FLAG_REG,0  ; Have 1st capture2
004A 0005          00186                RETFIE              ; Return from Interrupt
                   00187                                    ;
```

```
                00188  PAGE
004B 5422       00189 CAP2            MOVPF   CA2L,IC2BL  ; Move the captured value to
004C 5521       00190                 MOVPF   CA2H,IC2BH  ;   temporary registers
                00191                                     ;   (to prevent being overwritten)
                00192                                     ;
004D E061       00193                 CALL    SUB16       ; Call routine which subtracts
                00194                                     ;   2 16-bit numbers.
004E 9926       00195                 BTFSC   FLAG_REG,1  ; Underflow during SUB16?
004F 0725       00196                 DECF    T3OFLCNTR,1 ; Since underflow, decrement the
                00197                                     ;   overflow counter.
0050 2926       00198                 CLRF    FLAG_REG,1  ; Clear the flag bits for
                00199                                     ;   underflow and 2nd capture2
0051 6A25       00200                 MOVFP   T3OFLCNTR,W ; Store the T3 input capture
0052 4A20       00201                 MOVPF   W,IC2OF     ;   overflow value in IC2OF
0053 2825       00202                 CLRF    T3OFLCNTR,0 ; Clear the Data register which
                00203                                     ;   counts how many times Timer 3
                00204                                     ;   overflows.
0054 0005       00205                 RETFIE              ; Return from interrupt
                00206 ;
                00207 ; When Timer 3 has overflowed, the overflow counter only should
                00208 ; be incremented when the overflow occurs after a capture 1
                00209 ; but before the capture 2. The 4 possible cases when entering
                00210 ; the T3OVFL section of the PER_INT routine are as follows:
                00211 ;   Case 1: T3 overflow (only) and FLAG_REG.0 = 0 (waiting
                00212 ;           for Capture 1 to occur). Do Not increment counter
                00213 ;   Case 2: T3 overflow (only) and FLAG_REG.0 = 1 (waiting
                00214 ;           for Capture 2 to occur). Increment counter
                00215 ;   Case 3: T3 Overflow happened after Capture. Do Not
                00216 ;           increment overflow counter
                00217 ;   Case 4: T3 Overflow occured before Capture 2 and FLAG_REG.0 = 1
                00218 ;           (waiting for Capture 2 to occur). Increment counter
                00219
                00220 ;
0055 8E16       00221 T3OVFL          BCF     PIR,6       ; Clear Overflow interrupt flag
0056 9316       00222                 BTFSS   PIR,3       ; Did the RB1/CAP2 pin also
                00223                                     ;   cause an interrupt?
0057 C05E       00224                 GOTO    FR0         ; No, Check if between 1st
                00225                                     ;   and 2nd capture
0058 B803       00226                 MOVLB   3           ; Bank 3
0059 280A       00227                 CLRF    W,0         ; W = 0
005A 3115       00228                 CPFSEQ  CA2H        ; if CA2H = 0, overflow happened
005B C05E       00229                 GOTO    FR0         ;   first, must check FLAG_REG
                00230                                     ;   bit 0
005C B801       00231                 MOVLB   1           ; Back to bank 1
005D C043       00232                 GOTO    CAPTURE     ; Capture happened first, do NOT
                00233                                     ; Increment overflow counter
                00234                                     ;   and do capture routine
005E 9826       00235 FR0             BTFSC   FLAG_REG,0  ; Between Capture 1 and Capture 2?
005F 1525       00236                 INCF    T3OFLCNTR,1 ; Yes, Inc. the overflow counter
0060 0005       00237                 RETFIE              ; Return from overflow interrupt
                00238 ;
0061 6A24       00239 SUB16           MOVFP   IC2AL,W     ; Do the 16-bit subtraction
0062 0522       00240                 SUBWF   IC2BL,1     ;
0063 6A23       00241                 MOVFP   IC2AH,W     ;
0064 0321       00242                 SUBWFB  IC2BH,1     ;
0065 9004       00243                 BTFSS   ALUSTA,0    ; Is the result pos. or neg. ?
0066 8126       00244                 BSF     FLAG_REG,1  ; neg., Set the underflow flag
0067 0002       00245                 RETURN              ; Return from the subroutine
                00246  PAGE
                00247 ;
                00248 ; Other Interrupt routines. (Not utilized in this example)
                00249 ;
0068 0005       00250 EXT_INT RETFIE                      ; RA0/INT interrupt routine
                00251                                     ;   (NOT used in this program)
0069 0005       00252 TMR0INT RETFIE                      ; TMR0 overflow interrupt routine
                00253                                     ;   (NOT used in this program)
```

4

```
006A 0005           00254 T0INT  RETFIE                    ; RA1/T0CKI interrupt routine
                    00255                                  ;   (NOT used in this program)
                    00256                                  ;
006B C028           00257 SRESET  GOTO    START            ; If program became lost, goto
                    00258                                  ;   START and reinitialize.
                    00259 ;
                    00260 ;
                    00261 ; When the executed address is NOT in the program range, the
                    00262 ; 16-bit address should contain all 1's (a CALL 0x1FFF). At
                    00263 ; this location you could branch to a routine to recover or
                    00264 ; shut down from the invalid program execution.
                    00265 ;
07FF                00266                ORG             END_OF_PROG_MEM ;
07FF C06B           00267                GOTO    SRESET      ; The program has lost it's mind,
                    00268                                   ;   do a system reset
                    00269                END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-------X------- X-------X------- X-------XXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXX---- ----------------
07C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:    74


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: PERIOD MEASUREMENT, FREE RUNNING TIMER EXAMPLE CODE

```
MPASM 01.40 Released          IC_FRT2.ASM   1-16-1997  15:15:48         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 17C42, n = 66
                     00002 ;
                     00003 ;*****************************************************
                     00004 ;
                     00005 ; This is the basic outline for a program that can determine the
                     00006 ; frequency of an input, via input capture. This routine uses an
                     00007 ; 8-bit register to count the times that timer3 overflowed. At the
                     00008 ; Max crystal frequency of 16 MHz, this gives an overflow time of
                     00009 ; (2**16)(256 + 1)(250 nS) > 4.21 sec or a frequncy < 0.25 Hz. If
                     00010 ; measurement of longer time intervals is required, the overflow
                     00011 ; counter could be extended to 16 (or more) bits.
                     00012 ;
                     00013 ; Timer 3 in this example is a free running timer. The input
                     00014 ; capture is generated on the RB1/CAP2 pin. There is a flag
                     00015 ; that specifies if this is the 1st or 2nd capture.
                     00016 ; The first capture is the start of the period measurement. The
                     00017 ; second capture value gives the end of the period. In this type
                     00018 ; of measurement If the 2nd capture value < the 1st captue value
                     00019 ; then the overflow counter should be decremented.
                     00020 ;
                     00021 ; Do the EQUate table
                     00022 ;
  00000020           00023 IC2OF           EQU     0x20       ; T3 overflow register
  00000021           00024 IC2BH           EQU     0x21       ; T3 ICA2 MSB register (2nd Cap)
  00000022           00025 IC2BL           EQU     0x22       ; T3 ICA2 LSB register
  00000023           00026 IC2AH           EQU     0x23       ; T3 ICB2 MSB register (1st Cap)
  00000024           00027 IC2AL           EQU     0x24       ; T3 ICB2 LSB register
  00000025           00028 T3OFLCNTR       EQU     0x25       ; Temperay T3 overflow register
                     00029 ;
  00000026           00030 FLAG_REG        EQU     0x26       ; Register that has the Flag bits
                     00031 ;
                     00032 ;    FLAG_REG  bit   7   6   5   4   3   2   1   0
                     00033 ;                    -   -   -   -   -   -  UFL CAP1
                     00034 ;    CAP1 = 0,  1st Capture
                     00035 ;         = 1,  2nd Capture
                     00036 ;
                     00037 ;    UFL = 0,  No Underflow
                     00038 ;        = 1,  Underflow during subtract
                     00039 ;
                     00040 ;       Program:          IC_FRT2.ASM
                     00041 ;       Revision Date:
                     00042 ;                         1-14-97   Compatibility with MPASMWIN 1.40
                     00043 ;
                     00044 ;*****************************************************************
                     00045 ;
                     00046 ;
  000007FF           00047 END_OF_PROG_MEM EQU 0x07FF
                     00048 ;
                     00049 ;
  00000004           00050 ALUSTA          EQU     0x04
  00000006           00051 CPUSTA          EQU     0x06
  00000007           00052 INTSTA          EQU     0x07
  0000000A           00053 W               EQU     0x0A
```

```
                  00054 ;
  00000012        00055 PORTB           EQU     0x12        ; Bank 0
                  00056 ;
  00000016        00057 PIR             EQU     0x16        ; Bank 1
  00000017        00058 PIE             EQU     0x17
                  00059 ;
  00000012        00060 TMR3L           EQU     0x12        ; Bank 2
  00000013        00061 TMR3H           EQU     0x13
  00000016        00062 T3PRL           EQU     0x16
  00000017        00063 T3PRH           EQU     0x17
                  00064 ;
  00000014        00065 CA2L            EQU     0x14        ; Bank 3
  00000015        00066 CA2H            EQU     0x15
  00000016        00067 TCON1           EQU     0x16
  00000017        00068 TCON2           EQU     0x17
                  00069  PAGE
0000              00070                 ORG     0x0000      ; Origin for the RESET vector
0000 C028         00071                 GOTO    START       ; On reset, go to the start of
                  00072                                     ;   the program
0008              00073                 ORG     0x0008      ; Origin for the external RA0/INT
                  00074                                     ;   interrupt vector
0008 C068         00075                 GOTO    EXT_INT     ; Goto the ext. interrupt
                  00076                                     ;   on RA0/INT routine
0010              00077                 ORG     0x0010      ; Origin for the TMR0
                  00078                                     ;   overflow interrupt vector
0010 C069         00079                 GOTO    TMR0INT     ; Goto the TMR0 overflow interrupt
                  00080                                     ;   routine
0018              00081                 ORG     0x0018      ; Origin for the external
                  00082                                     ;   RB1/RT interrupt vector
0018 C06A         00083                 GOTO    T0INT       ; Goto the ext. interrupt on
                  00084                                     ;   RB1/RT routine
0020              00085                 ORG     0x0020      ; Origin for the interrupt vector
                  00086                                     ;   of any enabled peripheral
0020 C03E         00087                 GOTO    PER_INT     ; Goto the interrupt from a
                  00088                                     ;   peripheral routine
                  00089  PAGE
0028              00090                 ORG     0x0028      ; Origin for the top of
                  00091                                     ;   program memory
0028 8406         00092 START           BSF     CPUSTA,4    ; Disable ALL interrupts via the
                  00093                                     ;   Global Interrupt Disable
                  00094                                     ;   (GLINTD) bit.
                  00095                                     ;
0029              00096 MAIN                                ; Place Main program here
0029 B803         00097                 MOVLB   3           ; Select register Bank 3
002A 2817         00098                 CLRF    TCON2,0     ; Stop the timers, Single Capture
002B B070         00099                 MOVLW   0x070       ; Initialize TCON1 so that
002C 0116         00100                 MOVWF   TCON1       ;   T1 (8-bit), T2 (8-bit),
                  00101                                     ;   and T3 runs off the internal
                  00102                                     ;   system clock. Capture2
                  00103                                     ;   captures on the rising edge.
                  00104 ;
                  00105 ; Initialize Timer 3, load the timer with the number of cycles that
                  00106 ; the device executes (from RESET) before the timer is turned on
                  00107 ; Therefore the offset is required due to software overhead.
                  00108 ;
002D B802         00109                 MOVLB   2           ; Select register Bank 2
002E 280A         00110                 CLRF    W,0         ; Clear the W register
002F 0126         00111                 MOVWF   FLAG_REG    ; Initalize to 0
0030 0113         00112                 MOVWF   TMR3H       ; Timer3 MSB = 0
0031 B013         00113                 MOVLW   0x13        ; Timer3 LSB = Offset
0032 0112         00114                 MOVWF   TMR3L       ;
                  00115 ;
                  00116 ; Load the Timer 3 period register with 0xFFFF, which will give an
                  00117 ; interrupt on the overflow of Timer3
                  00118 ;
0033 B0FF         00119                 MOVLW   0xFF        ;
```


© 1997 Microchip Technology Inc.

```
0034 0117          00120                MOVWF   T3PRH       ;
0035 0116          00121                MOVWF   T3PRL       ;
                   00122 ;
                   00123 ; the timer should be started and interrupts enabled.
                   00124 ;
0036 B803          00125                MOVLB   3           ; Select register Bank 3
0037 8217          00126                BSF     TCON2,2     ; Turn on timer 3.
0038 8307          00127                BSF     INTSTA,3    ; Turn on Peripheral Interrupts
0039 B801          00128                MOVLB   1           ; Select register Bank 1
003A B048          00129                MOVLW   0x48        ; Enable Capture 2 and Timer3
003B 0117          00130                MOVWF   PIE         ; Interrupts (when GLINTD = 0)
                   00131 ;
                   00132 ; This is where you would do the things you wanted to do.
                   00133 ; this example will only loop waiting for the interrupts.
                   00134 ;
003C 8C06          00135 WAIT           BCF     CPUSTA,4    ; Enable ALL interrupts
003D C03C          00136                GOTO    WAIT        ; Loop here waiting for a timer
                   00137                                    ;   Interrupt
                   00138  PAGE
                   00139 ;
                   00140 ; The interrupt routine for any peripheral interrupt, This routine
                   00141 ; only deals with Timer3 (T3) interrupts.
                   00142 ;
                   00143 ; Time required to execute interrupt routine. Not including
                   00144 ; interrupt latency (time to enter into the interrupt routine)
                   00145 ;
                   00146 ;       case1 - only T3 overflow              = 12 cycles
                   00147 ;       case2 - 1st capture                   = 14 cycles
                   00148 ;       case3 - 2nd capture                   = 30 cycles
                   00149 ;       case4 - T3 overflow and 1st capture   = 34 cycles
                   00150 ;       case5 - T3 overflow and 2nd capture   = 50 cycles
                   00151 ;
                   00152 ;
003E B801          00153 PER_INT        MOVLB   1           ; Select register Bank 1
003F 9E16          00154                BTFSC   PIR,6       ; Did T3 overflow?
                   00155                                    ;   If not skip next Instruction
0040 C055          00156                GOTO    T3OVFL      ; Inc overflow cntr and clear flag
0041 9316          00157 CK_CAP         BTFSS   PIR,3       ; Did the RB1/CAP2 pin cause an
                   00158                                    ;   interrupt?
0042 0005          00159                RETFIE              ; No RB1/CAP2 interrupt,
                   00160                                    ;   Return from Interrupt
                   00161 ;
                   00162 ; This portion of the code takes the 1st capture and stores its
                   00163 ; value in register pair IC2AH:IC2AL. When the 2nd capture
                   00164 ; is taken, its value is stored in register pair IC2BH:IC2BL.
                   00165 ; A 16-bit subtract is performed, with the final 24-bit result
                   00166 ; being stored in IC2OF:IC2BH:IC2BL. This value will no longer
                   00167 ; be correct after the next capture occurs (IC2BH:IC2BL will
                   00168 ; change), so the main routine must utilize this value before
                   00169 ; it changes.
                   00170 ;
0043 8B16          00171 CAPTURE        BCF     PIR,3       ; Clear Capture2 interrupt flag
0044 B803          00172                MOVLB   3           ; Select register Bank 3
0045 9826          00173                BTFSC   FLAG_REG,0  ; 1st or 2nd capture2?
0046 C04B          00174                GOTO    CAP2        ; It was the 2nd Capture
0047 5424          00175 CAP1           MOVPF   CA2L,IC2AL  ; Move the captured value to
0048 5523          00176                MOVPF   CA2H,IC2AH  ;   temporary registers
0049 8026          00177                BSF     FLAG_REG,0  ; Have 1st capture2
004A 0005          00178                RETFIE              ; Return from Interrupt
                   00179                                    ;
                   00180  PAGE
004B 5422          00181 CAP2           MOVPF   CA2L,IC2BL  ; Move the captured value to
004C 5521          00182                MOVPF   CA2H,IC2BH  ;   temporary registers
                   00183                                    ;   (to prevent being overwritten)
                   00184                                    ;
004D E061          00185                CALL    SUB16       ; Call routine which subtracts
```


© 1997 Microchip Technology Inc.

```
                    00186                                   ;   2 16-bit numbers.
004E 9926           00187               BTFSC   FLAG_REG,1  ; Underflow during SUB16?
004F 0725           00188               DECF    T3OFLCNTR,1 ; Since underflow, decrement the
                    00189                                   ;   overflow counter.
0050 2926           00190               CLRF    FLAG_REG,1  ; Clear the flag bits for
                    00191                                   ;   underflow and 2nd capture2
0051 6A25           00192               MOVFP   T3OFLCNTR,W ; Store the T3 input capture
0052 4A20           00193               MOVPF   W,IC2OF     ;   overflow value in IC2OF
0053 2825           00194               CLRF    T3OFLCNTR,0 ; Clear the Data register which
                    00195                                   ;   counts how many times Timer 3
                    00196                                   ;   overflows.
0054 0005           00197               RETFIE              ; Return from interrupt
                    00198 ;
                    00199 ; When Timer 3 has overflowed, the overflow counter only should
                    00200 ; be incremented when the overflow occurs after a capture 1
                    00201 ; but before the capture 2. The 4 possible cases when entering
                    00202 ; the T3OVFL section of the PER_INT routine are as follows:
                    00203 ;   Case 1: T3 overflow (only) and FLAG_REG.0 = 0 (waiting
                    00204 ;           for Capture 1 to occur). Do Not increment counter
                    00205 ;   Case 2: T3 overflow (only) and FLAG_REG.0 = 1 (waiting
                    00206 ;           for Capture 2 to occur). Increment counter
                    00207 ;   Case 3: T3 Overflow happened after Capture. Do Not
                    00208 ;           increment overflow counter
                    00209 ;   Case 4: T3 Overflow occurred before Capture 2 and FLAG_REG.0 = 1
                    00210 ;           (waiting for Capture 2 to occur). Increment counter
                    00211
                    00212 ;
0055 8E16           00213 T3OVFL        BCF     PIR,6       ; Clear Overflow interrupt flag
0056 9316           00214               BTFSS   PIR,3       ; Did the RB1/CAP2 pin also
                    00215                                   ;   cause an interrupt?
0057 C05E           00216               GOTO    FR0         ; No, Check if between 1st
                    00217                                   ;   and 2nd capture
0058 B803           00218               MOVLB   3           ; Bank 3
0059 280A           00219               CLRF    W,0         ; W = 0
005A 3115           00220               CPFSEQ  CA2H        ; if CA2H = 0, overflow happened
005B C05E           00221               GOTO    FR0         ;   first, must check FLAG_REG
                    00222                                   ;   bit 0
005C B801           00223               MOVLB   1           ; Back to bank 1
005D C043           00224               GOTO    CAPTURE     ; Capture happened first, do NOT
                    00225                                   ;   Increment overflow counter
                    00226                                   ;   and do capture routine
005E 9826           00227 FR0           BTFSC   FLAG_REG,0  ; Between Capture 1 and Capture 2?
005F 1525           00228               INCF    T3OFLCNTR,1 ; Yes, Inc. the overflow counter
0060 0005           00229               RETFIE              ; Return from overflow interrupt
                    00230 ;
0061 6A24           00231 SUB16         MOVFP   IC2AL,W     ; Do the 16-bit subtraction
0062 0522           00232               SUBWF   IC2BL,1     ;
0063 6A23           00233               MOVFP   IC2AH,W     ;
0064 0321           00234               SUBWFB  IC2BH,1     ;
0065 9004           00235               BTFSS   ALUSTA,0    ; Is the result pos. or neg. ?
0066 8126           00236               BSF     FLAG_REG,1  ; neg., Set the underflow flag
0067 0002           00237               RETURN              ; Return from the subroutine
                    00238  PAGE
                    00239 ;
                    00240 ; Other Interrupt routines. (Not utilized in this example)
                    00241 ;
0068 0005           00242 EXT_INT RETFIE                    ; RA0/INT interrupt routine
                    00243                                   ;   (NOT used in this program)
0069 0005           00244 TMR0INT RETFIE                    ; TMR0 overflow interrupt routine
                    00245                                   ;   (NOT used in this program)
006A 0005           00246 T0INT   RETFIE                    ; RB1/RT interrupt routine
                    00247                                   ;   (NOT used in this program)
                    00248                                   ;
006B C028           00249 SRESET  GOTO    START             ; If program became lost, goto
                    00250                                   ;   START and reinitalize.
                    00251 ;
```


© 1997 Microchip Technology Inc.

```
                    00252 ;
                    00253 ; When the executed address is NOT in the program range, the
                    00254 ; 16-bit address should contain all 1's (a CALL 0x1FFF). At
                    00255 ; this location you could branch to a routine to recover or
                    00256 ; shut down from the invalid program execution.
                    00257 ;
07FF                00258                   ORG             END_OF_PROG_MEM ;
07FF C06B           00259                   GOTO    SRESET      ; The program has lost it's mind,
                    00260                                       ;   do a system reset
                    00261                   END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-------X------- X-------X------- X-------XXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXX---- ----------------
07C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:    74


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

4

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C: PULSE WIDTH MEASUREMENT EXAMPLE CODE

```
MPASM 01.40 Released              PW02.ASM   1-16-1997  15:16:15         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 17C42, n = 66
                     00002 ;
                     00003 ;*****************************************************
                     00004 ;
                     00005 ; This is the basic outline for a program that can determine the
                     00006 ; Pulse Width of an input, via input capture. This routine uses an
                     00007 ; 8-bit register to count the times that timer3 overflowed. At the
                     00008 ; Max crystal frequency of 16 MHz, this gives an overflow time of
                     00009 ; (2**16)(256 + 1)(250 nS) > 4.21 sec or a frequency < 0.25 Hz. If
                     00010 ; measurement of longer time intervals is required, the overflow
                     00011 ; counter could be extended to 16 (or more) bits.
                     00012 ;
                     00013 ; Do the EQUate table
                     00014 ;
00000020             00015 IC2OF        EQU     0x20        ; T3 overflow register
00000021             00016 IC2BH        EQU     0x21        ; T3 ICA2 MSB register (2nd Cap)
00000022             00017 IC2BL        EQU     0x22        ; T3 ICA2 LSB register
00000023             00018 IC2AH        EQU     0x23        ; T3 ICB2 MSB register (1st Cap)
00000024             00019 IC2AL        EQU     0x24        ; T3 ICB2 LSB register
00000025             00020 T3OFLCNTR    EQU     0x25        ; Temperature T3 overflow register
                     00021 ;
00000026             00022 FLAG_REG        EQU 0x26         ; Register that has the Flag bits
                     00023 ;
                     00024 ;    FLAG_REG  bit   7   6   5   4   3   2   1   0
                     00025 ;                    -   -   -   -   -   -  UFL CAP1
                     00026 ;    CAP1 = 0,  1st Capture
                     00027 ;         = 1,  2nd Capture
                     00028 ;
                     00029 ;    UFL = 0,  No Underflow
                     00030 ;        = 1,  Underflow during subtract
                     00031 ;
                     00032 ;       Program:          PW02.ASM
                     00033 ;       Revision Date:
                     00034 ;                         1-14-97   Compatibility with MPASMWIN 1.40
                     00035 ;
                     00036 ;*************************************************************************
                     00037 ;
                     00038 ;
000007FF             00039 END_OF_PROG_MEM EQU     0x07FF
                     00040 ;
00000004             00041 ALUSTA          EQU     0x04
00000006             00042 CPUSTA          EQU     0x06
00000007             00043 INTSTA          EQU     0x07
0000000A             00044 W               EQU     0x0A
                     00045                                 ;
00000012             00046 PORTB           EQU     0x12        ; Bank 0
                     00047                                 ;
00000016             00048 PIR             EQU     0x16        ; Bank 1
00000017             00049 PIE             EQU     0x17
                     00050                                 ;
00000012             00051 TMR3L           EQU     0x12        ; Bank 2
00000013             00052 TMR3H           EQU     0x13
00000016             00053 T3PRL           EQU     0x16
00000017             00054 T3PRH           EQU     0x17
```


© 1997 Microchip Technology Inc.

```
                    00055                                   ;
  00000014          00056 CA2L        EQU     0x14        ; Bank 3
  00000015          00057 CA2H        EQU     0x15
  00000016          00058 TCON1       EQU     0x16
  00000017          00059 TCON2       EQU     0x17
                    00060  PAGE
                    00061 ;
0000                00062             ORG     0x0000      ; Origin for the RESET vector
0000 C028           00063             GOTO    START       ; On reset, go to the start of
                    00064                                 ;   the program
0008                00065             ORG     0x0008      ; Origin for the external RA0/INT
                    00066                                 ;   interrupt vector
0008 C072           00067             GOTO    EXT_INT     ; Goto the ext. interrupt
                    00068                                 ;   on RA0/INT routine
0010                00069             ORG     0x0010      ; Origin for the TMR0
                    00070                                 ;   overflow interrupt vector
0010 C073           00071             GOTO    TMR0INT     ; Goto the TMR0 overflow interrupt
                    00072                                 ;   routine
0018                00073             ORG    0x0018       ; Origin for the external
                    00074                                 ;   RA1/T0CKI interrupt vector
0018 C074           00075             GOTO    T0CKI_INT   ; Goto the ext. interrupt on
                    00076                                 ;   RA1/T0CKI routine
0020                00077             ORG     0x0020      ; Origin for the interrupt vector
                    00078                                 ;   of any enabled peripheral
0020 C03E           00079             GOTO    PER_INT     ; Goto the interrupt from a
                    00080                                 ;   peripheral routine
0028                00081             ORG     0x0028      ; Origin for the top of
                    00082                                 ;   program memory
                    00083  PAGE
0028 8406           00084 START       BSF     CPUSTA,4    ; Disable ALL interrupts via the
                    00085                                 ;   Global Interrupt Disable
                    00086                                 ;   (GLINTD) bit.
                    00087                                 ;
0029                00088 MAIN                            ; Place Main program here
0029 B803           00089             MOVLB   3           ; Select register Bank 3
002A 2817           00090             CLRF    TCON2,0     ; Stop the timers, Single Capture
002B B070           00091             MOVLW   0x070       ; Initialize TCON1 so that
002C 0116           00092             MOVWF   TCON1       ;   T1 (8-bit), T2 (8-bit),
                    00093                                 ;   and T3 run off the internal
                    00094                                 ;   system clock. Capture2
                    00095                                 ;   captures on the rising edge.
                    00096 ;
                    00097 ; Initialize Timer 3, load the timer with the number of cycles that
                    00098 ; the device executes (from RESET) before the timer is turned on
                    00099 ; Therefore the offset is required due to software overhead.
                    00100 ;
002D B802           00101             MOVLB   2           ; Select register Bank 2
002E 280A           00102             CLRF    W,0         ; Clear the W register
002F 0126           00103             MOVWF   FLAG_REG    ; Initialize to 0
0030 0113           00104             MOVWF   TMR3H       ; Timer3 MSB = 0
0031 B013           00105             MOVLW   0x13        ; Timer3 LSB = Offset
0032 0112           00106             MOVWF   TMR3L       ;
                    00107 ;
                    00108 ; Load the Timer 3 period register with 0xFFFF, which will give an
                    00109 ; interrupt on the overflow of Timer3
                    00110 ;
0033 B0FF           00111             MOVLW   0xFF        ;
0034 0117           00112             MOVWF   T3PRH       ;
0035 0116           00113             MOVWF   T3PRL       ;
                    00114 ;
                    00115 ; the timer should be started and interrupts enabled.
                    00116 ;
0036 B803           00117             MOVLB   3           ; Select register Bank 3
0037 8217           0011              BSF     TCON2,2     ; Turn on timer 3.
0038 8307           00119             BSF     INTSTA,3    ; Turn on Peripheral Interrupts
0039 B801           00120             MOVLB   1           ; Select register Bank 1
```

4

```
003A B048          00121              MOVLW   0x48        ; Enable Capture 2 and Timer3
003B 0117          00122              MOVWF   PIE         ; Interrupts (when GLINTD = 0)
                   00123 ;
                   00124 ; This is where you would do the things you wanted to do.
                   00125 ; this example will only loop waiting for the interrupts.
                   00126 ;
003C 8C06          00127 WAIT         BCF     CPUSTA,4    ; Enable ALL interrupts
003D C03C          00128              GOTO    WAIT        ; Loop here waiting for a timer
                   00129                                  ;   Interrupt
                   00130  PAGE
                   00131 ;
                   00132 ; The interrupt routine for any peripheral interrupt, This routine
                   00133 ; only deals with Timer3 (T3) interrupts.
                   00134 ;
                   00135 ; Time required to execute interrupt routine. Not including
                   00136 ; interrupt latency (time to enter into the interrupt routine)
                   00137 ;
                   00138 ;       case1 - only T3 overflow             = 12 cycles
                   00139 ;       case2 - 1st capture                  = 20 cycles
                   00140 ;       case3 - 2nd capture                  = 34 cycles
                   00141 ;       case4 - T3 overflow and 1st capture  = 32 cycles
                   00142 ;       case5 - T3 overflow and 2nd capture  = 44 cycles
                   00143 ;
                   00144 ;
003E B801          00145 PER_INT      MOVLB   1           ; Select register Bank 1
003F 9E16          00146              BTFSC   PIR,6       ; Did T3 overflow?
                   00147                                  ;   If not skip next Instruction
0040 C05A          00148              GOTO    T3OVFL      ; Inc overflow cntr and clear flag
0041 9316          00149 CK_CAP       BTFSS   PIR,3       ; Did the RB1/CAP2 pin cause an
                   00150                                  ;   interrupt?
0042 0005          00151              RETFIE              ; No RB1/CAP2 interrupt,
                   00152                                  ;   Return from Interrupt
                   00153
                   00154 ;
                   00155 ; This portion of the code takes the 1st capture and stores its
                   00156 ; value in register pair IC2AH:IC2AL. When the 2nd capture
                   00157 ; is take, its value is stored in register pair IC2BH:IC2BL.
                   00158 ; A 16-bit subtract is performed, with the final 24-bit result
                   00159 ; being stored in IC2OF:IC2BH:IC2BL. This value will no longer
                   00160 ; be correct after the next capture occurs (IC2BH:IC2BL will
                   00161 ; change), so the main routine must utilize this value before
                   00162 ; it changes.
                   00163 ;
0043 B803          00164 CAPTURE      MOVLB   3           ; Select register Bank 3
0044 9826          00165              BTFSC   FLAG_REG,0  ; Capture on rising or falling edge?
0045 C04B          00166              GOTO    FALLING     ; It was the 2nd Capture
0046 5424          00167 RISING       MOVPF   CA2L,IC2AL  ; Move the captured value to
0047 5523          00168              MOVPF   CA2H,IC2AH  ;   temporary registers
0048 8026          00169              BSF     FLAG_REG,0  ; Set flag for 1st capture
0049 8E16          00170              BCF     TCON1,6     ; Change edge from rising
                   00171                                  ;   to falling
004A C055          00172              GOTO    FALSE_C     ;** With the changing of the capture
                   00173                                  ;** edge, we have a false capture
                   00174                                  ;
                   00175  PAGE
004B 5422          00176 FALLING      MOVPF   CA2L,IC2BL  ; Move the captured value to
004C 5521          00177              MOVPF   CA2H,IC2BH  ;   temporary registers
                   00178                                  ;   (to prevent being overwritten)
                   00179                                  ;
004D E06B          00180              CALL    SUB16       ; Call routine which subtracts
                   00181                                  ;   2 16-bit numbers.
004E 9926          00182              BTFSC   FLAG_REG,1  ; Underflow during SUB16?
004F 0725          00183              DECF    T3OFLCNTR,1; Since underflow, decrement the
                   00184                                  ;   overflow counter.
0050 2926          00185              CLRF    FLAG_REG,1  ; Clear the flag bits for
                   00186                                  ;   underflow and 2nd capture2
```


© 1997 Microchip Technology Inc.

```
0051 6A25          00187              MOVFP   T3OFLCNTR,W; Store the T3 input capture
0052 4A20          00188              MOVPF   W,IC2OF    ;   overflow value in IC2OF
0053 2825          00189              CLRF    T3OFLCNTR,0; Clear the Data register which
                   00190                                 ;   counts how many times Timer 3
                   00191                                 ;   overflows.
0054 8616          00192              BSF     TCON1,6    ; Change edge from falling
                   00193                                 ;   to rising
                   00194 ;
                   00195 ; Note when you change the edge of the capture, an additional capture
                   00196 ; is generated. The capture register must be read before the capture
                   00197 ; flag is cleared.
                   00198 ;
0055 550A          00199 FALSE_C      MOVPF   CA2H,W     ; Dummy read of Capture 2
0056 540A          00200              MOVPF   CA2L,W     ;
                   00201 ;
0057 B801          00202              MOVLB   1          ; Select register Bank 1
0058 8B16          00203              BCF     PIR,3      ; Clear Capture2 Interrupt flag
0059 0005          00204              RETFIE             ; Return from interrupt, wait for
                   00205                                 ;   T3 overflow or falling edge
                   00206                                 ;   capture
                   00207                                 ;
                   00208  PAGE
                   00209 ;
                   00210 ; When Timer 3 has overflowed, the overflow counter only should
                   00211 ; be incremented when the overflow occurs after a capture 1
                   00212 ; but before the capture 2. The 6 possible cases when entering
                   00213 ; the T3OVFL section of the PER_INT routine are as follows:
                   00214 ;
                   00215 ;   Case 1: T3 overflow (only) and FLAG_REG.0 = 0 (waiting
                   00216 ;           for Capture 1 to occur). Do Not increment counter
                   00217 ;   Case 2: T3 overflow (only) and FLAG_REG.0 = 1 (waiting
                   00218 ;           for Capture 2 to occur). Increment counter
                   00219 ;   Case 3: T3 Overflow, Then Capture1 happened. Do Not
                   00220 ;           increment overflow counter
                   00221 ;   Case 4: T3 Overflow, Then Capture 2 happened
                   00222 ;           Increment counter
                   00223 ;   Case 5: Capture1, Then T3 Overflow happened
                   00224 ;           Increment counter
                   00225 ;   Case 6: Capture2, Then T3 Overflow happened. Do Not
                   00226 ;           Increment counter
                   00227 ;
005A 8E16          00228 T3OVFL       BCF     PIR,6      ; Clear Overflow interrupt flag
005B 9316          00229              BTFSS   PIR,3      ; Did the RB1/CAP2 pin also
                   00230                                 ;   cause an interrupt?
005C C068          00231              GOTO    FR0        ; No, only overflow interrupt
005D B803          00232              MOVLB   3          ; Bank 3
005E 280A          00233              CLRF    W,0        ; W = 0
005F 9826          00234              BTFSC   FLAG_REG,0 ; T3 overflow with Capture 1
                   00235                                 ;   or Capture 2?
0060 C064          00236              GOTO    OF_C1      ; Overflow with Capture 1
0061 3115          00237 OF_C2        CPFSEQ  CA2H       ; if CA2H = 0, overflow happened
                   00238                                 ;   first
0062 1525          00239              INCF    T3OFLCNTR,1; Increment counter
0063 C043          00240              GOTO    CAPTURE    ; Do capture routine
0064 3115          00241 OF_C1        CPFSEQ  CA2H       ; if CA2H = 0, overflow happened
                   00242                                 ;   first
0065 C043          00243              GOTO    CAPTURE    ; Capture happened first, do NOT
                   00244                                 ;   Increment overflow counter
                   00245                                 ;   and do capture routine
0066 1525          00246              INCF    T3OFLCNTR,1; Yes, Inc. the overflow counter
0067 C043          00247              GOTO    CAPTURE    ; Do capture routine
                   00248 ;
                   00249 ; Only increment overflow counter if between 1st and 2nd capture
                   00250 ;
0068 9826          00251 FR0          BTFSC   FLAG_REG,0 ; Between Capture 1 and Capture 2?
0069 1525          00252              INCF    T3OFLCNTR,1; Yes, Inc. the overflow counter
```


© 1997 Microchip Technology Inc.

```
006A 0005           00253              RETFIE              ; Return from overflow interrupt
                    00254 ;
006B 6A24           00255 SUB16        MOVFP   IC2AL,W     ; Do the 16-bit subtraction
006C 0522           00256              SUBWF   IC2BL,1     ;
006D 6A23           00257              MOVFP   IC2AH,W     ;
006E 0321           00258              SUBWFB  IC2BH,1     ;
006F 9004           00259              BTFSS   ALUSTA,0    ; Is the result pos. or neg. ?
0070 8126           00260              BSF     FLAG_REG,1  ; neg., Set the underflow flag
0071 0002           00261                 RETURN           ; Return from the subroutine
                    00262  PAGE
                    00263 ;
                    00264 ; Other Interrupt routines. (Not utilized in this example)
                    00265 ;
0072 0005           00266 EXT_INT RETFIE                   ; RA0/INT interrupt routine
                    00267                                  ;   (NOT used in this program)
0073 0005           00268 TMR0INT RETFIE                   ; TMR0 overflow interrupt routine
                    00269                                  ;   (NOT used in this program)
0074 0005           00270 T0INT  RETFIE                    ; RA1/T0CKI interrupt routine
                    00271                                  ;   (NOT used in this program)
                    00272                                  ;
0075 C028           00273 SRESET  GOTO    'START           ; If program became lost, goto
                    00274                                  ;   START and reinitalize.
                    00275 ;
                    00276 ;
                    00277 ; When the executed address is NOT in the program range, the
                    00278 ; 16-bit address should contain all 1's (CALL 0x1FFF). At
                    00279 ; this location you could branch to a routine to recover or
                    00280 ; shut down from the invalid program execution.
                    00281 ;
07FF                00282          ORG     END_OF_PROG_MEM;
07FF C075           00283          GOTO    SRESET          ; The program has lost it's mind,
                    00284                                  ;   do a system reset
                    00285                  END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X-------X------- X-------X------- X-------XXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXX----------
07C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:    84


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Serial Port Utilities

Author: *Amar Palacherla*
Microchip Technology Inc.

INTRODUCTION

The PIC17C42 has an on-chip high speed Universal Synchronous Asynchronous Receiver Transmitter (USART). The serial port can be configured to operate either in full-duplex asynchronous mode or half duplex synchronous mode. The serial port has a dedicated 8-bit baud rate generator. Either 8- or 9-bits can be transmitted/received.

This application note provides information on using the serial port, parity generation, serial port expansion, RS-232 interface, I/O port expansion using the serial port in synchronous mode.

SERIAL PORT USAGE

A brief code to setup the serial port, and receive and transmit data is given in Example 1. Small sections of code for both asynchronous and synchronous mode are given.

Asynchronous Mode Setup

Asynchronous mode setup requires selection of 8/9-bits of data transfer, baud rate, setting the baud rate generator, and configuring the TXSTA and RCSTA control registers. The baud rate generator is configured by writing the appropriate value to SPBRG register (bank0, file 17h). The value to be written to SPBRG is given by:

$$SPBRG = \frac{Input_Clk_Freq}{64 \bullet Baud_Rate} - 1$$

For example, to select a baud rate of 9600 bits/sec with input clock frequency of 16 MHz, SPBRG is computed from the above equation to be 25. Once the Baud Rate Generator is set up, it is necessary to configure the TXSTA and RCSTA control registers as shown in Example 1 (refer to the data sheet):

4

EXAMPLE 1: INITIALIZATION EXAMPLE

```
SPBRG   :   25                  :   9600 baud @ 16 MHz input clock
TXSTA   :   00100000    (20h)   :   8-bit transmission, async mode
RCSTA   :   10010000    (90h)   :   8-bit reception, enable serial port, enable reception

;**********************************************************************
;       Sample Code For Asynchronous Mode Serial Port Setup
;**********************************************************************

#define ClkFreq         16000000                    ; input clock frequency = 16 MHz
#define baud(X)         ((10*ClkFreq/(64*X))+5)/10 - 1
#define TXSTA_INIT      0x90
#define RCSTA_INIT      0x90

    #include            "17C42.h"                   ; file containing the Register Definitions

    Setup_Async_Mode
                movlb   0                           ; SPBRG, TXSTA & RCSTA are in bank 0
                movlw   baud(9600)                  ; equals 25 for 9600 baud
                movwf   SPBRG                       ; baud rate generator is reset & initialized
                movlw   TXSTA_INIT
                movwf   TXSTA                       ; 8-bit transmission, async mode
                movlw   RCSTA_INIT
                movwf   RCSTA                       ; 8-bit reception, enable serial port,
                                                    ; enable reception
                return
;**********************************************************************
```

© 1997 Microchip Technology Inc.

Synchronous Mode Setup

Synchronous mode setup requires selection of 8/9-bits of data transfer, bit rate, setting the baud rate generator, and configuring the TXSTA and RCSTA control registers. The baud rate generator is configured by writing the appropriate value to SPBRG register (bank0, file 17h). The value to be written to SPBRG is given by:

$$SPBRG = \frac{Input_Clk_Freq}{4 \bullet Baud_Rate} - 1$$

For example, to select a bit rate of 1 Mbits/sec with input clock frequency of 16 MHz, SPBRG is computed from the above equation to be 3. Once the Baud Rate Generator is set up, it is necessary to configure the TXSTA and RCSTA control registers as follows (please refer to the data sheet) :

EXAMPLE 2: CONFIGURING TXSTA AND RCSTA CONTROL REGISTORS

```
SPBRG   :   3                   :   1 Mbits/sec @ 16 Mhz input clock
TXSTA   :   10110000    (B0h)   :   8-bit transmission, Sync mode (MASTER)
RCSTA   :   10010000    (90h)   :   8-bit reception, enable serial port, continuous
                                    reception

;**********************************************************************
;          Sample Code For Synchronous Mode (MASTER) Serial Port Setup
;**********************************************************************

#define ClkFreq         16000000                ; input clock frequency = 16 Mhz
#define baud(X)         ((10*ClkFreq/(4*X))+5)/10 - 1
#define TXSTA_INIT      0xB0
#define RCTSA_INIT      0x90

        #include                "17C42.h"           ; file containing the Register Definitions

Setup_Sync_Master_Mode
                movlb   0                       ; SPBRG, TXSTA & RCSTA are in bank 0
                movlw   baud(1000000)           ; equals 3 for 1 Mbits/sec
                movwf   SPBRG                   ; baud rate generator is reset & initialized
                movlw   TXSTA_INIT
                movwf   TXSTA                   ; 8-bit transmission, async mode
                movlw   RCSTA_INIT
                movwf   RCSTA                   ; 8-bit reception, enable serial port,
                                                ; enable reception
                return
;**********************************************************************
```


© 1997 Microchip Technology Inc.

Receiving Data (Software Polling)

The sample code in Example 3 provides a way to read the received serial data by software polling (with no serial port interrupts). This applies to both asynchronous and synchronous mode. Software polling is done by checking the RCIF bit (PIR<0>). If this bit is set it means that a word has been received (8 bits are in RCREG and the 9th bit in RCSTA<0>).

EXAMPLE 3: POLLING FOR RECEIVE DATA

```
;*********************************************************************
;          Return The 8-bit received Data By Software Polling
;       The received data is returned in location SerInData
;*********************************************************************
Get_Serial_Data_Poll
               movlb   1                ; PIR is in bank 1
PollRcv        btfss   PIR,0            ; check the RBIF bit
               goto    PollRcv          ; loop until char received, assume WDT is off
               movlb   0                ; RCREG is in bank 0
               movpf   RCREG,SerInData
               return                   ; Received 8-bits are in SerInData
;*********************************************************************
```

Transmitting Data (Software Polling)

The sample code in Example 4 provides a way to transmit serial data by software polling (no serial port interrupts). Software polling is done by checking the bit (PIR<1> in bank 1) to be one, indicating the transfer of TXIF to the serial shift register.

EXAMPLE 4: POLLING FOR TRANSMIT DATA

```
;*********************************************************************
;          Transmit 8-bit  Data By Software Polling
;       The data to be transmitted is in location SerOutData
;*********************************************************************
Send_Serial_Data_Poll
               movlb   1                ; PIR is in bank 1
PollTXIFbtfss  PIR,1                    ; check the TXIF bit of PIR register in bank1
               goto    PollTXIF         ; loop until char received, assume WDT is off
               movlb   0                ; RCREG is in bank 0
               movfp   SerOutData,TXREG
               return                   ; Received 8-bits are in SerInData
;*********************************************************************
```


© 1997 Microchip Technology Inc.

Transmitting & Receiving A Block Of Data (Interrupt Driven)

A general purpose routine which is interrupt driven, that transmits and receives a block of data is provided in Example 5. The reception or transmission of the block is ended when an end of block character is detected. As an example, the end of block is identified by a 0. The block of data to be transmitted is stored in the program memory and the TABLRD instruction is used to transfer this example data to the file registers and serial port. The user may modify this code to a more general purpose routine that suits the application.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

EXAMPLE 5: INTERRUPT DRIVEN TRANSMIT/RECEIVE

```
MPASM 01.40 Released           SERINT.ASM   1-22-1997  10:20:49         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;         TITLE   'Serial Interface Routines
                     00002 ;         PROCESSOR     42
                     00003
                     00004 ;This is a short program to demonstrate how to transmit and receive
                     00005 ;serial data using the PIC17C42.
                     00006 ;
                     00007 ;A message will be transmitted and routed right back to the processor
                     00008 ;and read. The read information will be saved in an internal buffer.
                     00009 ;
                     00010 ;       Program:          SERINT.ASM
                     00011 ;       Revision Date:
                     00012 ;                       1-22-97      Compatibility with MPASMWIN 1.40
                     00013 ;
                     00014 ;
                     00015         LIST     P = 17C42
                     00016
                     00017 #include    <p17c42.inc>
                     00001         LIST
                     00002 ;P17C42.INC Standard Header File, Version 1.03 Microchip Technology, Inc.
                     00264         LIST
                     00018
  00000080           00019 TX_BUFFER       equ     0x80
  000000B0           00020 RX_BUFFER       equ     0xB0
  00000020           00021 RXPTR           equ     0x20
  00000021           00022 TXPTR           equ     0x21
  00000022           00023 SERFLAG         equ     0x22
  00000023           00024 RTINUM          equ     0x23
  00000001           00025 RXDONE          equ     1
  00000000           00026 TXDONE          equ     0
  00000002           00027 HILOB           equ     2
                     00028 ;
                     00029 ;
0000                 00030         ORG     0
0000 C072            00031         goto    start
                     00032 ;
0010                 00033         ORG     0x0010          ;vector for tmr0 interrupt
                     00034 ;tmr0_int                       ;not used here
                     00035 ;
0020                 00036         ORG     0x0020          ;vector for peripheral interrupt
0020                 00037 perf_int
0020 C04D            00038         goto    service_perf    ;service the interrupts
                     00039 ;
0030                 00040         ORG     0x0030
                     00041 ;
                     00042 ;initialize the serial port baud rate interrupts etc.
```


© 1997 Microchip Technology Inc.

```
0030                00043 init_serial
0030 2922           00044         clrf    SERFLAG, F      ;clear all flags
0031 B800           00045         movlb   0               ;select bank 0
0032 B007           00046         movlw   0x07            ;select 9600 baud
0033 7700           00047         movfp   W,SPBRG         ;       /
0034 B090           00048         movlw   0x90            ;set up serial pins
0035 7300           00049         movfp   W,RCSTA         ;       /
0036 2915           00050         clrf    TXSTA, F        ;setup transmit status
0037 B801           00051         movlb   1               ;select bank 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0038 2916           00052         clrf    PIR, F          ;clear all interrupts
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0039 2917           00053         clrf    PIE, F          ;clear all enables
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
003A 8017           00054         bsf     PIE,RCIE        ;enable receive interrupt
003B B0B0           00055         movlw   RX_BUFFER       ;set pointer to rx buffer
003C 4020           00056         movpf   W,RXPTR         ;       /
003D 2907           00057         clrf    INTSTA, F       ;clear all interrupts
003E 8307           00058         bsf     INTSTA,PEIE     ;enable peripheral ints
003F 0005           00059         retfie
                    00060 ;
                    00061 ;start transmission of first two bytes
0040                00062 start_xmit
0040 B800           00063         movlb   0               ;select bank 0
0041 8515           00064         bsf     TXSTA,TXEN      ;enable transmit
0042 AB00           00065         tablrd  1,1,W           ;load latch
0043 A216           00066         tlrd    1,TXREG         ;load high byte
0044 B801           00067         movlb   1               ;select bank 1
0045                00068 empty_chk
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0045 9116           00069         btfss   PIR,TXIF        ;TXBUF empty?
0046 C045           00070         goto    empty_chk       ;no then keep checking
0047 B800           00071         movlb   0               ;select bank 0
0048 A916           00072         tablrd  0,1,TXREG       ;load lo byte
0049 B801           00073         movlb   1               ;select bank 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
004A 8117           00074         bsf     PIE,TXIE        ;enable transmit interrupts
004B 8222           00075         bsf     SERFLAG,HILOB   ;set up next for high byte
004C 0002           00076         return
                    00077 ;
                    00078 ;
                    00079         PAGE
                    00080 ;
004D                00081 service_perf
                    00082 ;check for transmit or receive interrupts only
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
004D 9816           00083         btfsc   PIR,RCIF        ;RX buffer full?
004E C062           00084         goto    service_recv    ;yes then service
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
004F 9116           00085         btfss   PIR,TXIF        ;TX buffer empty?
0050 C060           00086         goto    exit_perf       ;no, ignore other int.
0051                00087 service_xmt
0051 9822           00088         btfsc   SERFLAG,TXDONE  ;all done?
0052 C060           00089         goto    exit_perf       ;yes then quit
0053 9A22           00090         btfsc   SERFLAG,HILOB   ;if clr, do low byte
0054 C057           00091         goto    rd_hi           ;else read high byte
0055 A900           00092         tablrd  0,1,W           ;read lo
0056 C058           00093         goto    sx_cont         ;continue
0057                00094 rd_hi
0057 A200           00095         tlrd    1,W             ;read high byte
0058                00096 sx_cont
0058 3A22           00097         btg     SERFLAG,HILOB   ;toggle flag
0059 B800           00098         movlb   0               ;bsr=0
005A 4016           00099         movpf   W,TXREG         ;load tx reg
005B 3300           00100         tstfsz  W               ;last byte?
005C C060           00101         goto    exit_perf       ;no then cont
```


© 1997 Microchip Technology Inc.

```
005D                00102 end_xmt                         ;else end transmit
005D B801           00103         movlb   1               ;select bank 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
005E 8917           00104         bcf     PIE,TXIE        ;disable tx interrupt
005F 8022           00105         bsf     SERFLAG,TXDONE  ;set done flag
0060                00106 exit_perf
0060 8F07           00107         bcf     INTSTA,PEIF     ;clear peripheral int
0061 0005           00108         retfie
                    00109 ;
0062                00110 service_recv
0062 9922           00111         btfsc   SERFLAG,RXDONE  ;RX complete?
0063 C060           00112         goto    exit_perf       ;exit int
0064 6120           00113         movfp   RXPTR,FSR0      ;get pointer
0065 B800           00114         movlb   0               ;select bank 0
0066 6014           00115         movfp   RCREG,INDF0     ;load received value
0067 290A           00116         clrf    WREG, F         ;clr W
0068 3200           00117         cpfsgt  INDF0           ;value = 0?
0069 C06D           00118         goto    end_recv        ;yes then end
006A 1501           00119         incf    FSR0, F         ;inc pointer
006B 4120           00120         movpf   FSR0,RXPTR      ;save pointer
006C C060           00121         goto    exit_perf       ;return from int
006D                00122 end_recv
006D 8122           00123         bsf     SERFLAG,RXDONE  ;set flag
006E 2907           00124         clrf    INTSTA, F       ;clear all int
006F B801           00125         movlb   1               ;select bank 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0070 8817           00126         bcf     PIE,RCIE        ;disable rx interrupts
0071 C060           00127         goto    exit_perf       ;return
                    00128         PAGE
                    00129
                    00130 ;
0072                00131 start
0072 2909           00132         clrf    FSR1, F         ;assign FSR1 as S.P.
0073 0709           00133         decf    FSR1, F         ;     /
0074 B020           00134         movlw   0x20            ;clear ram space
0075 6100           00135         movfp   W,FSR0          ;do indirect addressing
0076                00136 start1
0076 2900           00137         clrf    INDF0, F        ;clear ram
0077 1F01           00138         incfsz  FSR0, F         ;inc and skip if done
0078 C076           00139         goto    start1
0079 E030           00140         call    init_serial     ;initialize serial port
007A B000           00141         movlw   LOW   MESSAGE   ;load table pointer
007B 400D           00142         movpf   W,TBLPTRL       ;     /
007C B001           00143         movlw   HIGH  MESSAGE   ;     /
007D 400E           00144         movpf   W,TBLPTRH       ;     /
007E E040           00145         call    start_xmit      ;start transmission
007F                00146 chk_end
007F 9122           00147         btfss   SERFLAG,RXDONE  ;receive all?
0080 C07F           00148         goto    chk_end         ;no then keep checking
                    00149 ;
0081 C081           00150 loop    goto    loop            ;spin wheel
                    00151 ;
0100                00152         ORG     0x100
0100                00153 MESSAGE
0100 5468 6520 636F 00154         DATA    "The code is: Tea for the Tillerman"
     6465 2069 733A
     2054 6561 2066
     6F72 2074 6865
     2054 696C 6C65
     726D 616E
0111 0000           00155         DATA    0
                    00156 ;
                    00157 ;
                    00158         END
```


© 1997 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X--------------- ---------------- X--------------- XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XX-------------- ---------------- ---------------- ----------------
0100 : XXXXXXXXXXXXXXXX XX-------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:   102


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     9 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

PARITY GENERATION

Since the serial port of the PIC17C42 does not have an on-chip parity generator, parity is generated using software. It takes only 10 program memory words and executes in 10 instruction cycles to generate parity. Since the serial port of the PIC17C42 can operate in a 9-bit mode, the parity bit can be generated in software and transmitted as the 9th bit or it can be compared with the received 9th bit.

In case of transmission, set TX9 to 1 (TXSTA < 6>) to enable 9-bit transmission and write the computed parity bit to TX9D (TXSTA<0>). The 9th bit (parity bit) must be written prior to writing the 8 data bits to TXREG.

In case of a reception, first of all enable 9-bit reception by setting RX9 to 1 (RCSTA<6>). Upon successful reception, the 9th bit is received in RX9D (RCSTA<0>). Parity of the 8 bits of received data is computed using the routine listed below and compared with the 9-bit received.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

EXAMPLE 6: PRIORITY GENERATION

```
MPASM 01.40 Released           PARITY.ASM   1-22-1997  10:17:25         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         TITLE   "Generate Parity Bit"
                     00002 ;
                    00003 ;*************************************************************************
                     00004 ;       Generate Parity Bit for the 8 bit register 'txmt'
                     00005 ;       The parity bit is stored in Bit 0 of 'parity'
                     00006 ;
                     00007 ;
                     00008 ;       Program:          PARITY.ASM
                     00009 ;       Revision Date:
                     00010 ;                         1-22-97  Compatibility with MPASMWIN 1.40
                     00011 ;
                    00012 ;*************************************************************************
                     00013 ;
                     00014         LIST            P = 17C42
                     00015
                     00016         #include        <p17c42.inc>
                     00001         LIST
                    00002 ;P17C42.INC Standard Header File, Version 1.03 Microchip Technology, Inc.
                     00264         LIST
                     00017
                     00018         #define TRUE    1
                     00019         #define FALSE   0
                     00020
                     00021 #define ODD_PARITY      FALSE
                     00022 ;
                     00023
                     00024         CBLOCK  0x20
  00000020           00025                 txmt, parity
                     00026         ENDC
                     00027
0000 1C20            00028         swapf   txmt,W
0001 0C20            00029         xorwf   txmt,W
0002 0121            00030         movwf   parity
0003 2121            00031         rrncf   parity, F
0004 2121            00032         rrncf   parity, F
0005 0C21            00033         xorwf   parity,W
0006 B503            00034         andlw   0x03
0007 B101            00035         addlw   0x01
0008 210A            00036         rrncf   WREG, F
0009 0121            00037         movwf   parity
```


© 1997 Microchip Technology Inc.

```
                    00038 ;
                    00039  #if ODD_PARITY
                    00040          btg     parity,0
                    00041  #endif
                    00042 ;
                    00043          END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXX------ ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:    10

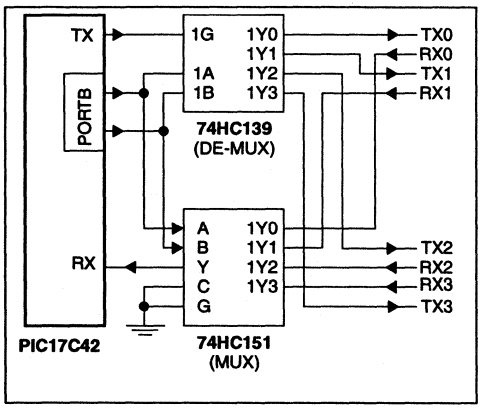

Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

SERIAL PORT EXPANSION

The PIC17C42 has only one serial port. For applications that require the PIC17C42 to communicate with multiple serial ports, a scheme that multiplexes and demultiplexes the RX and TX pins is provided below. This method is suitable only if no more than one USART is needed at any one time. This is the case in many applications where the microcontroller drives several output devices serially. Figure 1, shown below suggests a way to expand the on-chip serial port to four serial ports. To use the scheme shown in Figure 1, The PIC17C42 must select the desired serial port by appropriately setting the two pins of PORTB. The same scheme may be used to further expand the serial ports by using more I/O Ports.

FIGURE 1: MULTIPLEXING THE ON-CHIP USART



RS-232 INTERFACE

Two circuits are provided to interface the CMOS levels of a PIC17C42 to the RS-232 levels. Figure 2 provides an interface to MAX232 (MAXIM's RS-232 Driver/Receiver) with a single +5V power supply. Figure 3 provides a low cost two-chip solution for the RS-232 level translation using a single +5V supply (Note that V- of MC14C88 is connected to DTR of RS-232 Interface. By asserting DTR to low, V- gets the negative voltage from the RS-232 line). An alternative single chip low cost solution is provided in Figure 4. However, 3V sources (+5, +12, -12) are necessary.

FIGURE 2: RS-232 INTERFACE TO MAX232

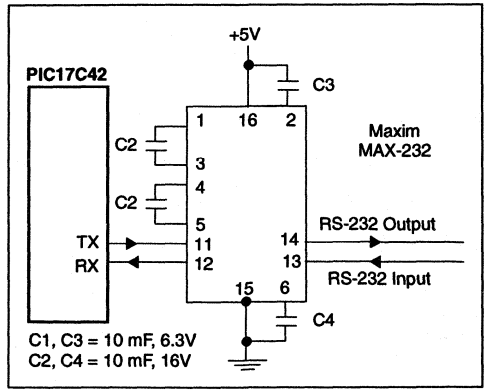


FIGURE 3: LOW-COST TWO-CHIP SOLUTION USING SINGLE POWER SOURCE

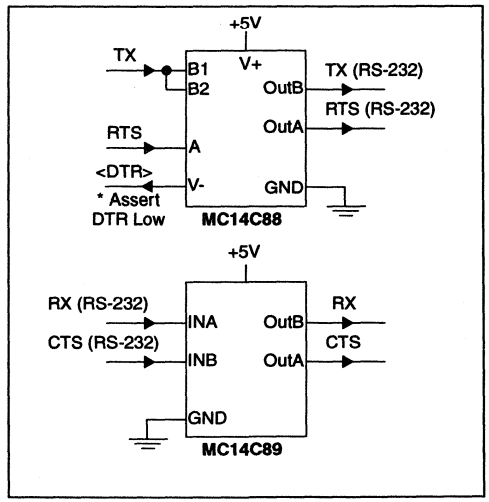
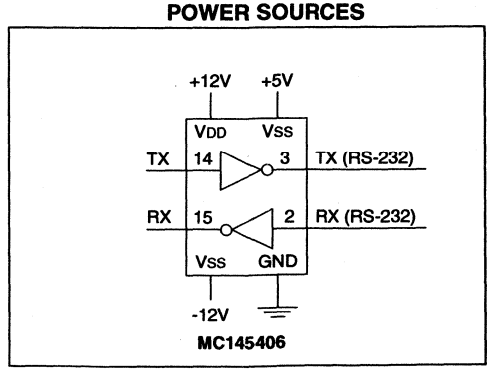


FIGURE 4: LOW-COST SINGLE CHIP SOLUTION USING THREE-POWER SOURCES



© 1997 Microchip Technology Inc.

Table 1 provides the summary of RS-232 and V.28 Electrical Specifications.

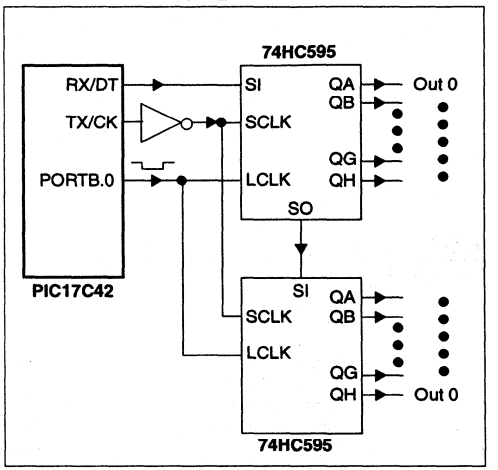
TABLE 1: SUMMARY OF RS-232C AND V.28 ELECTRICAL SPECIFICATIONS

Parameter	Specification	Comments
Driver Output Voltage		
0 level 1 level Max. output	+5V to +15V -5V to -15V ±25V	With 3-7 kΩ load With 3-7 kΩ load No load
Receiver Input Thresholds (Data and clock signals)		
0 level 1 level	+3V to +125V -3V to -25V	
Receiver Thresholds RTS, DSR, DTR		
On level Off level	+3V to +25V Open circuit or -3V to -25V	Detects power Off condition at driver
Receiver Input resistance		
	3 kΩ to 7 kΩ	
Driver Output resistance		
Power off condition Driver slew rate	300Ω Min. 30V/µs max.	VOUT < ±2V 3 kΩ < RL < 7k 0pF < CL < 2500 pF
Signalling rate		
	Up to 20K bits/sec.	
Cable length		
	50'/15m. Recommended max. length.	Longer cables permissible, if CLOAD ≤ 2500 pF

I/O PORT EXPANSION USING SYNCHRONOUS MODE

Although the PIC17C42 has 33 I/O pins, most of these are multiplexed with other peripheral functions. In cases where more I/O ports are needed, the scheme below expands the I/O port using the synchronous mode of serial port by serially shifting the data. Figure 5 shows a scheme to expand the output ports to 16 bits using two standard logic chips (74HC595). The PIC17C42's serial port is configured in synchronous mode and set to be the MASTER. Thus serial data is available on DT (pin 22) and the clock is available on CK (pin 21). The following code will transmit 16 bits serially and clock all the 16 bits at the same time.

FIGURE 5: OUTPUT PORT EXPANSION USING SYNCHRONOUS MODE



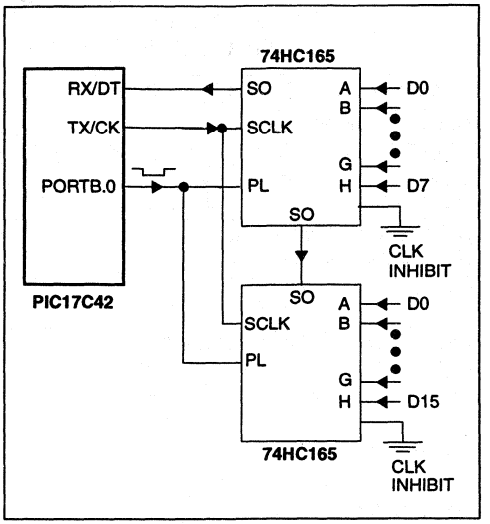
EXAMPLE 7: PORT IMPORT EXPANSION

```
InitSerialPortTxmt
        movlb   0
        clrf    SPBRG                   ; set to highest baud rate = CLKOUT = CLKIN/4
        movlw   0x80
        movwf   RCSTA                   ; enable serial port
        movlw   0xB0
        movwf   TXSTA                   ; 8 bit synchronous master mode
        bcf     DDRB,0                  ; set bit 0 of PortB as output, to be used as Latch Clk
        return
SendSerialData                          ; shift out DataLo & DataHi serially
        movlb   0
        movfp   DataLo,TXREG
        nop                             ; wait for TXREG transfer : if slower baud rate is
                                        ; used check for TBMT reset
        movfp   DataHi,TXREG
wait    btfss   TXSTA,TRMT              ; wait until all 16 bits shifted out
        goto    wait
        bcf     PortB,0
        bsf     PortB,0                 ; clock in the serial data to parallel output of HC595
        return
```


© 1997 Microchip Technology Inc.

A similar scheme as shown above may be implemented in a reverse manner to expand the input ports. This requires a Parallel In and Serial Out device. Using a standard logic chip (74HC165), the scheme is shown in Figure 6. In order to read the 16-bit input data, the serial port of the PIC17C42 is configured for Synchronous Mode Reception (MASTER mode). An I/O port (PORTB<0>) is used to parallel load the 16 inputs for reading serially. A sample code to read the 16 inputs is shown below.

FIGURE 6: INPUT PORT EXPANSION USING SYNCHRONOUS MODE



EXAMPLE 8:

```
InitSerialPortRcv
            movlb   0
            clrf    SPBRG               ; set to highest baud rate = CLKOUT = CLKIN/4
            movlw   0x80
            movwf   RCSTA               ; enable serial port
            movlw   0x90
            movwf   TXSTA               ; 8-bit synchronous master mode
            bcf     DDRB,0              ; bit 0 of PortB is output, to be used as Parallel
                                        ; Load
            bsf     PortB,0             ; disable parallel Load
            return
ReadSerialData                          ; shift out DataLo & DataHi serially
            movlb   0
            bcf     PortB,0             ; Parallel Load The Inputs into 74HC165
            bsf     PortB,0             ; disable parallel Load
            bsf     RCSTA,SREN          ; enable single byte reception and wait for data
            movlb   1
    wait1   btfss   PIR,RBIF
            goto    wait1               ; check until 8-bits are received
            movlb   0
            movpf   RCREG,DataLo        ; 1st byte is read
            movlb   0
            bsf     RCSTA,SREN          ; enable another byte of reception and wait for data
            movlb   1
    wait2   btfss   PIR,RBIF
            goto    wait2               ; check until 8-bits are received
            movlb   0
            movpf   RCREG,DataHi        ; 2nd byte is read
            return
```


© 1997 Microchip Technology Inc.

Implementing Table Read and Table Write

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

This application note discusses how to retrieve data from program memory to data memory and write data from data memory to program memory.

RETLW K Instruction

As in all PIC17CXXX family parts, the simplest method used to retrieve data from program memory to data memory is to use the RETLW K instruction. For example:

```
; simple program to transfer
; table values to PortB
Main
        movlw   5,W         ;load offset
        call    SimpleTableRead
        movwf   PortB       ;output to PortB
        •
        •
        •
SimpleTableRead
        addwf   PC          ;add offset to PC
Table   retlw   0           ;return a known
                            ;table value based
                            ;on the OFFSET.
        •
        •
        •
        retlw   10
```

In the example above, OFFSET is loaded with the required offset to the Table and the subroutine SimpleTableRead is called. The table value is returned in the W register. In this manner program memory can be transferred to data memory.

Table Read Instruction

The PIC17C42 has an expanded instruction set which includes the TABLRD and TLRD instructions. These instructions are specifically constructed to transfer data from program memory to data memory.

If the instruction syntax is: TABLRD t,i,f, the sequence in which this instruction is executed is as follows:

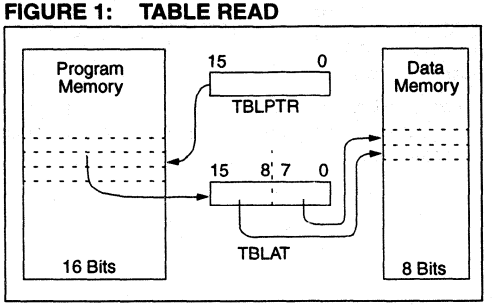
a) if t = 1 then the high byte of the table latch (TBLATH) is loaded in the file register f.
b) else (if t = 0) the low byte of the table latch (TBLATL) is loaded in the file register f.
c) next, the 16 bit data pointed to by the table pointer (TBLPTR) is loaded into the table latch.
d) lastly, if i = 1 the table pointer (TBLPTR) is incremented.

Note: The first time this instruction is executed in a sequence, the table latch will not be initialized, hence an unknown value will be loaded in the file register. This is not a problem if the user overwrites the same f register in the next subsequent instruction.

If the instruction syntax is: TLRD t,f, the sequence in which this instruction is executed is as follows:

a) if t = 1 then the high byte of the table latch (TBLATH) is loaded in the file register f.
b) else (if t = 0) the low byte of the table latch (TBLATH) is loaded in the file register f.

FIGURE 1: TABLE READ



4

© 1997 Microchip Technology Inc.

Read In-Line

A simple method of transferring data from program memory to data memory is to use the TABLRD and TLRD instructions in sequence as shown in the example below:

```
;transfer 6 bytes of data from program memory
;at 0x500, to data memory at 0x80
ReadInLine
   movlw   05          ;load table pointer
                       ; with 0x500
   movwf   TBLPTRH     ;       /
   clrf    TBLPTRL     ;       /
   tablrd  0,1,0x80    ;get 16 bit value in
                       ;table latch.
   tlrd    0,0x80      ;low byte (1st) @ 80
   tablrd  1,1,0x81    ;high byte (2nd) @ 81
   tlrd    0,0x82      ;3rd byte @ 82
   tablrd  1,1,0x83    ;4th byte @ 83
   tlrd    0,0x84      ;5th byte @ 84
   tablrd  1,1,0x85    ;6th byte @ 85
```

Reading a Block of Data

In instances where a block of N bytes needs to be transferred from program memory to data memory, the TABLRD and TLRD instruction need to be included in a loop which checks for N transfers.

```
;transfer 'COUNT' bytes (even values only) of
;data at program memory 'MESSAGE' to data
;memory at: 'RAM_BUFFER'
ReadBlock
   movlw   high MESSAGE    ;load table pointer
   movpf   W,TBLPTRH       ;       /
   movlw   low MESSAGE     ;       /
   movpf   W,TBLPTRL       ;       /
   bcf     ALUSTA,5        ;enable post auto
                           ;increment of FSR0
   movlw   RAM_BUFFER      ;initialize FSR0
                           ;to RAM_BUFFER
   movfp   W,FSR0          ;       /
   movlw   COUNT/2         ;initialize count
   tablrd  1,1,RAM_BUFFER  ;initialize table
                           ;latch
ReadBlockLoop
   tlrd    1,0x00          ;do indirect read
                           ;of high byte
   tablrd  0,1,0x00        ;do indirect read
                           ;of low byte
   decfsz  W               ;check if count=0
   goto    ReadBlockLoop   ;no then do next
   return                  ;else end of
                           ;transfer.
```

Program	Code Size	Transfer Rate
Simple Table Read (using RETLW)	N + 3	6 cycles/byte
Read In-Line	4 + N + N/2	1.5 cycles/byte
Read Block (using loop)	14 + N/2	3 cycles/byte

N = Number of bytes to transfer

	Code Size		
	Simple Table Read	Read In-Line	Read Block
N = 10	13	19	19
N = 20	23	34	24

Conclusion:

In cases where the number of bytes to be transferred is small, the Read In-Line offers small code size for fast transfer rate. However, as the number of bytes to be transferred increases, the Read Block offers optimum code size for a decent transfer rate.

© 1997 Microchip Technology Inc.

Table Write Instruction

The PIC17C42 has a TABLWT and a TLWT instruction which transfer data from data memory to program memory. Note in cases where the table pointer points to internal EPROM, the table write instruction will try to program the EPROM, hence the programming voltage must be present on the VPP line to successfully program the part.

The instruction syntax is: TABLWT t,i,f.

The sequence in which this instruction is executed is as follows:

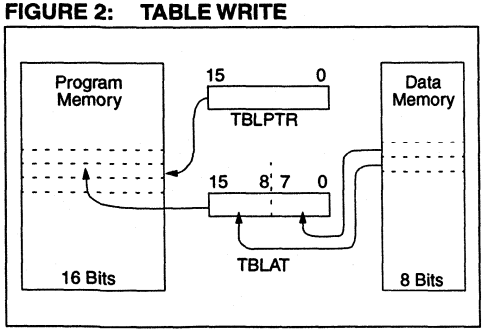
a) if t = 1 then the file register f is loaded to the high byte of the table latch (TBLATH).
b) else (if t = 0) the file register f is loaded to the low byte of the table latch (TBLATL).
c) next, the 16-bit data in the table latch is transferred to the program memory pointed to by the table pointer (TBLPTR).
d) lastly, if i = 1 the table pointer (TBLPTR) is incremented.

The instruction syntax is: TLWT t,f

The sequence in which this instruction is executed is as follows:

a) if t = 1 then the file register f is loaded to the high byte of the table latch (TBLATH).
b) else (if t = 0) the file register f is loaded to the low byte of the table latch (TBLATL).

FIGURE 2: TABLE WRITE



Write In-Line

A simple method of transferring data from data memory to program memory is to use the TABLWT and TLWT instruction in sequence as shown in the example below:

```
;transfer 6 bytes of data in data memory at
;0x80, to program memory at 0x5000:
ReadInLine
   movlw   50          ;load table pointer
                       ;with 0x5000
   movwf   TBLPTRH     ;       /
   clrf    TBLPTRL     ;       /
   tlwt    1,0x80      ;high byte @ table latch.
   tablwt  0,1,0x81    ;low byte @ table latch
                       ;latch  @ prog. mem.
   tlwt    1,0x82      ;3rd and 4th byte
                       ;@ prog. mem.
   tablwt  0,1,0x83    ;       /
   tlwt    1,0x84      ;5th and 6th byte
                       ;@ prog. mem.
   tablwt  0,1,0x85    ;       /
```

Writing a Block of Data

In instances where a block of N bytes needs to be transferred from data memory to program memory, the tablwt and tlwt instructions need to be included in a loop which checks for N transfers.

```
;transfer 'COUNT' bytes (even values only) of
;data at program memory at 'RAM_BUFFER' to
;program memory at 'MESSAGE'
WriteBlock
   movlw   high MESSAGE    ;load table pointer
   movpf   W,TBLPTRH       ;       /
   movlw   low MESSAGE     ;       /
   movpf   W,TBLPTRL       ;       /
   bcf     ALUSTA,5        ;enable post auto
                           ;increment of FSR0
   movlw   RAM_BUFFER      ;initialize FSR0
                           ;to RAM_BUFFER
   movfp   W,FSR0          ;       /
   movlw   COUNT/2         ;initialize count
WriteBlockLoop
   tlwt    1,0x00          ;high byte
                           ;@ table latch
   tablwt  0,1,0x00        ;low byte @ table
                           ;latch;
                           ;table latch
                           ;@ prog. mem.
   decfsz  W               ;check if count = 0
   goto    WriteBlockLoop  ;no then do next
   return                  ;else end of transfer
```

4

© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Using the PWM

Author: *Mark Palmer*
Microchip Technology Inc.

INTRODUCTION

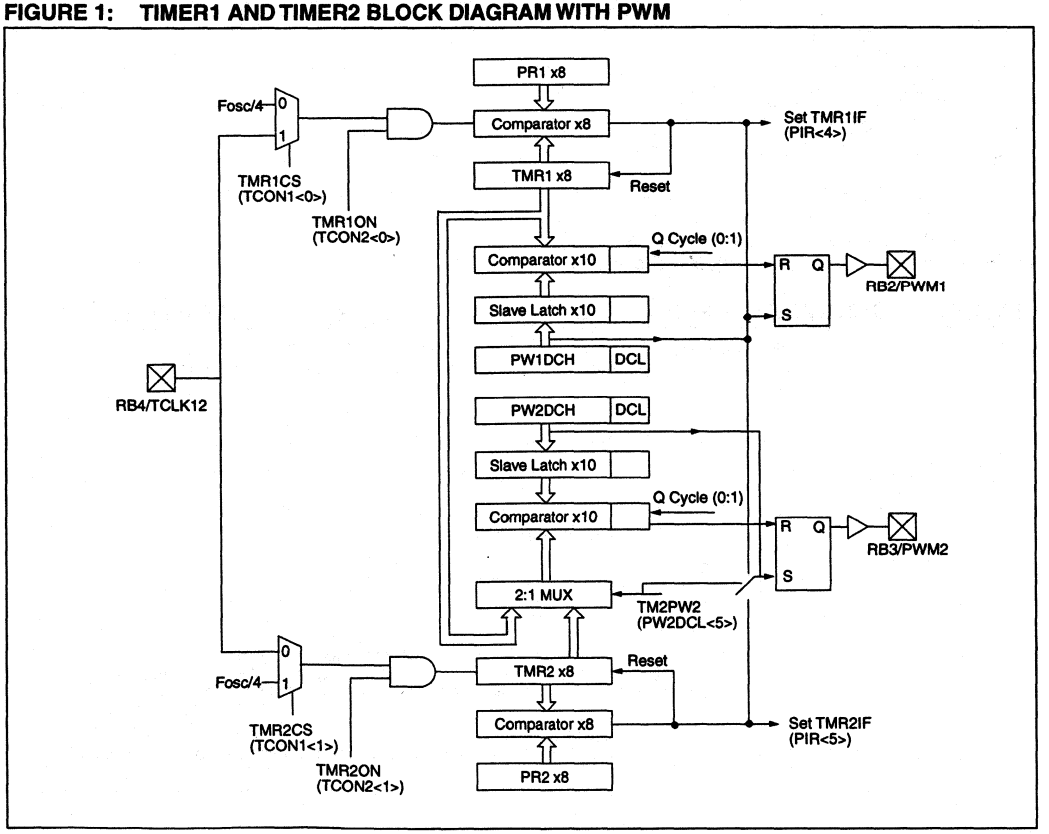
The PICmico™ family of RISC microcontrollers hax been designed to provide advanced performance and a cost-effective solution for a variety of applications. This application note provides examples which illustrate some uses of Pulse Width Modulation (PWM) using the PIC17C42's Timer1 or Timer2 module. These examples may be modified to suit the specific needs of your application.

This Application Note describes the operation of the PWM. They include the following topics:

1. Simple PWM Operation
2. Variable Period / Variable Duty Cycle PWM
3. External Clock for Timer Time-base (ramifications/issues)

The listing file for the Variable Period / Variable Duty Cycle example can be found in Appendix A. The source files can be found on the Microchip BBS. On directions on how to access the Microchip BBS please refer to DS30128, which can also be found in the Microchip Embedded Control Handbook (Literature Number DS00092).

FIGURE 1: TIMER1 AND TIMER2 BLOCK DIAGRAM WITH PWM



© 1997 Microchip Technology Inc.

Control registers that are used by Timer1 and Timer2 are shown in Table 1. Shaded Boxes are control bits that are not used by Timer1 or Timer2.

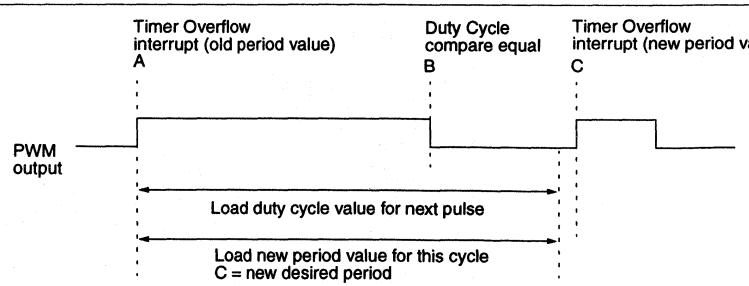
TABLE 1: REGISTERS ASSOCIATED WITH TIMER3 AND CAPTURE

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-On Reset	Value on all other resets (Note1)
16h, Bank 3	TCON1	CA2ED1	CA2ED0	CA1ED1	CA1ED0	T16	TMR3CS	TMR2CS	TMR1CS	0000 0000	0000 0000
17h, Bank 3	TCON2	CA2OVF	CA1OVF	PWM2ON	PWM1ON	CA1/PR3	TMR3ON	TMR2ON	TMR1ON	0000 0000	0000 0000
10h, Bank 2	TMR1	Timer1								xxxx xxxx	uuuu uuuu
11h, Bank 2	TMR2	Timer2								xxxx xxxx	uuuu uuuu
12h, Bank 2	TMR3L	Timer3 low byte								xxxx xxxx	uuuu uuuu
13h, Bank 2	TMR3H	Timer3 high byte								xxxx xxxx	uuuu uuuu
16h, Bank 1	PIR	RBIF	TMR3IF	TMR2IF	TMR1IF	CA2IF	CA1IF	TXIF	RCIF	0000 0010	0000 0010
17h, Bank 1	PIE	RBIE	TMR3IE	TMR2IE	TMR1IE	CA2IE	CA1IE	TXIE	RCIE	0000 0000	0000 0000
07h, Unbanked	INTSTA	PEIF	T0CKIF	T0IF	INTF	PEIE	T0CKIE	T0IE	INTE	0000 0000	0000 0000
06h, Unbanked	CPUSTA	—	—	STKAV	GLINTD	TO	PD	—	—	--11 11--	--11 ??--
14h, Bank 2	PR1	Timer1 period register								xxxx xxxx	uuuu uuuu
15h, Bank 2	PR2	Timer2 period register								xxxx xxxx	uuuu uuuu
16h, Bank 2	PR3L/CA1L	Timer3 period register, low byte/capture1 register, low byte								xxxx xxxx	uuuu uuuu
17h, Bank 2	PR3H/CA1H	Timer3 period register, high byte/capture1 register, high byte								xxxx xxxx	uuuu uuuu
10h, Bank 3	PW1DCL	DC1	DC0	—	—	—	—	—	—	xx-- ----	uu-- ----
11h, Bank 3	PW2DCL	DC1	DC0	TM2PW2	—	—	—	—	—	xx0- ----	uu0- ----
12h, Bank 3	PW1DCH	DC9	DC8	DC7	DC6	DC5	DC4	DC3	DC2	xxxx xxxx	uuuu uuuu
13h, Bank 3	PW2DCH	DC9	DC8	DC7	DC6	DC5	DC4	DC3	DC2	xxxx xxxx	uuuu uuuu
14h, Bank 3	CA2L	Capture2 low byte								xxxx xxxx	uuuu uuuu
15h, Bank 3	CA2H	Capture2 high byte								xxxx xxxx	uuuu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented, reads as '0'. ? - Value depends on condition.
Note 1: Shaded cells are not used by TMR1, TMR2,TMR3 or Capture.

Care must be taken when loading values into the PWM registers. These registers are the duty cycle registers (PWxDCH:PWxDCL) and the period register (PRx). Figure 2 shows proper update timing of these values.

FIGURE 2: TIMING FOR UPDATING THE DUTY CYCLE REGISTERS AND PERIOD REGISTER



Note: When updating the period register, the value loaded must be greater than the timer value. If the period value is less than the timer value, the duty cycle value is not latched and the timer is not reset to 00h until the next Timer = Period match. This causes the current cycle to not output as expected and cause a "glitch".

Note: It is generally good practice to load the new period value into the PRx register as soon as possible after Point A.

© 1997 Microchip Technology Inc.

SIMPLE PWM OPERATION

Simple PWM operation is where the period of the PWM output remains constant, and only the duty cycle is modified. The PWM can operate in either of two modes:

- Hi-resolution mode: the PWxDCL register is modified
- Standard resolution mode: the PWxDCL register is not modified

When operating in the standard-resolution mode, only the PWDCH register is ever modified. Since the modification takes only a single cycle and can be done at any time. Also since the period is remaining constant this may be done without any PWM interrupt software overhead.

When operating in the high-resolution mode both the PWxDCH:PWxDCL register pair is modified. Since this is a multicycle update, care needs to be taken that the "new" PWM duty cycle value is not latched until the update is complete. If the duty cycle is latched before this update is complete, the duty cycle will display a "glitch". If the PWxDCH is written first, the maximum error is 3 Q-cycles (187.5 ns @ 16 MHz). If the PWxDCL is written first, the maximum error is also 3 Q-cycles (187.5 ns @ 16 MHz), with the PWxDCH delayed by one PWM period. This may be acceptable for some applications. If this is not acceptable for your application then a subroutine can be written to ensure that these duty cycle writes are not done when the timer will equal the period. One implementation of this subroutine (`PWM_UD`) is used in the Variable Period / Variable Duty Cycle PWM example. This is discussed in the following section, with the listing in Appendix A.

Additional code examples can be found in application note AN539 in the Embedded Control Handbook.

VARIABLE PERIOD / VARIABLE DUTY CYCLE PWM

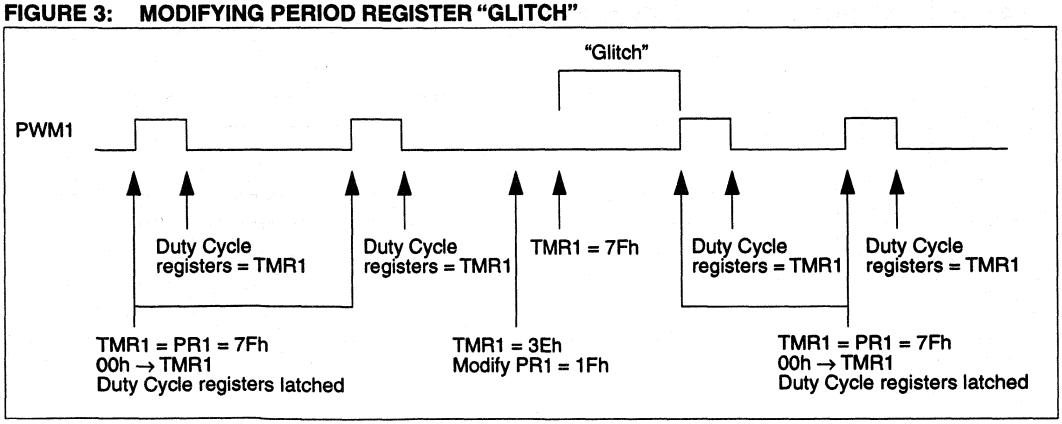
In a variable period / variable duty cycle PWM both the duty cycle of the PWM as well as the frequency (period) of the PWM are modified.

The PIC17C42's hardware double buffers the duty cycle registers, but the period registers are not double buffered. What this means is that you can modify the duty cycle registers, but the value will only be latched when the timer register equals the period register. Since the period register is not buffered, as the period register is modified this becomes the "new" period. This means that care must be taken when modifying the period register. The most common problem would be to modify the period register resulting in a "glitch." This "glitch" occurs when the period register is modified with a value that is less than the present timer value. The timer does not have a match with the old period value, and continues to count until the timer register equals the period register.

Figure 3, shows an example where PR1 the register period = 7Fh. Then the period is modified to a smaller value (PR1 = 1Fh) without checking that the value in Timer1 (TMR1) register = 3Eh. Since the new period (PR1) value is less then the present timer (TMR1) value, a glitch has occurred.

4

FIGURE 3: MODIFYING PERIOD REGISTER "GLITCH"



© 1997 Microchip Technology Inc.

Care must be taken when writing a 10-bit duty cycle value. Since this requires two register writes, the "Timer Equals Period" could occur between these two writes, which would give a duty cycle that was not as expected. The cases are as follows:

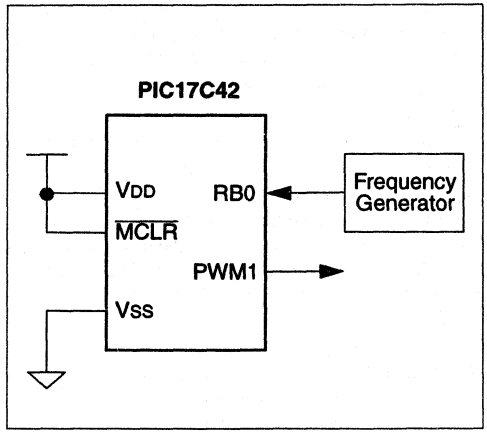
a) If the duty cycle low register (DCL) is written, and then the Timer equals period. The old DCH register and the new DCL register becomes the duty cycle.

b) If the duty cycle high register (DCH) is written, and then the Timer equals period. The new DCH register and the old DCL register becomes the duty cycle

At the following occurrence of the timer equaling the period, the second register written would be updated. The subroutine `PWM_UD` (Appendix A) ensures that these duty cycle writes are not done when the timer will equal the period.

A software example of a variable period / variable duty cycle is shown in Appendix A. In this example the period is double buffered in software, and the new period value is loaded in the timer overflow interrupt service routine. When the new duty cycle needs to be loaded. The device connections are shown in Figure 4. This program has two PWM settings (period / duty cycle combinations) that are switched between depending on the level on pin RB0. A frequency generator was used to give a low frequency signal on the RB0 pin. Figure 5 shows an example of the input and output waveforms.

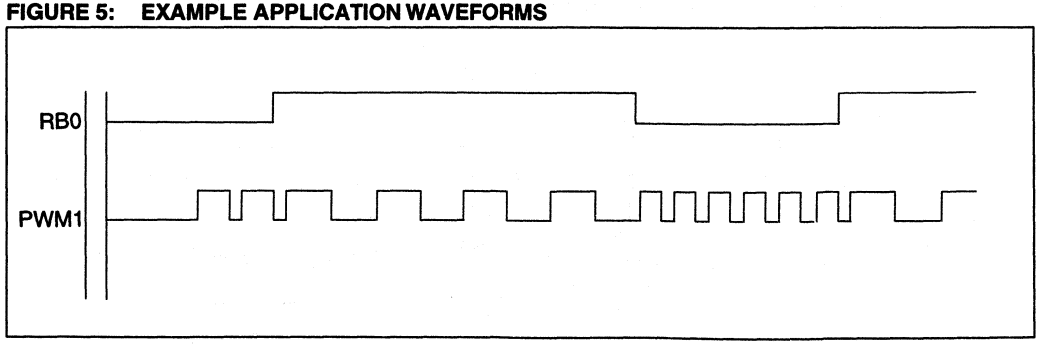
FIGURE 4: APPLICATION HARDWARE SETUP



The program listing in Appendix A implements this example, Figure 8 is the hardware function. This example may be modified to suit the particular needs of your application. The following table is a summary of the requirements for this program (@ 16 MHz):

Code Size:	52 Words
RAM used:	11 Bytes
Interrupt Service Routine time	3.0 µs
Subroutine time	4.5 µs 6.0 µs
Maximum PWM frequency:	200 kHz
PWM Accuracy:	62.5 µs

FIGURE 5: EXAMPLE APPLICATION WAVEFORMS



© 1997 Microchip Technology Inc.

EXTERNAL CLOCK FOR TIMER TIMEBASE

The counters used for the time-base of the PWM outputs can be software selected to operate from an external clock source. This allows a lower frequency PWM to be achieved. This brings up new issues that must be understood for the application.

One of these issues is clock synchronization. All external clocks must be synchronized to the internal operating speed of the microcontroller, as shown in Figure 6. When this synchronization occurs the PWM output is not truly operating from the external clock, but actually the internal synchronized clock. This leads to a "jitter" of the output to the clock. This jitter is caused from the delta time between the external clock and the synchronized clock not being constant. The synchronization errors are:

Duty cycle error = ± TCY

Period error = ± TCY

If you needed to run the PWM at a low frequency, and also want to reduce the "jitter" from the use of an external asynchronous clock, a PWM output could be used as the synchronous clock source. When the clock is synchronized to the device the clock error is always constant, so there is no jitter. Figure 7 shows this example.

FIGURE 7: PWM OUTPUT TO GENERATE A SYNCHRONOUS CLOCK

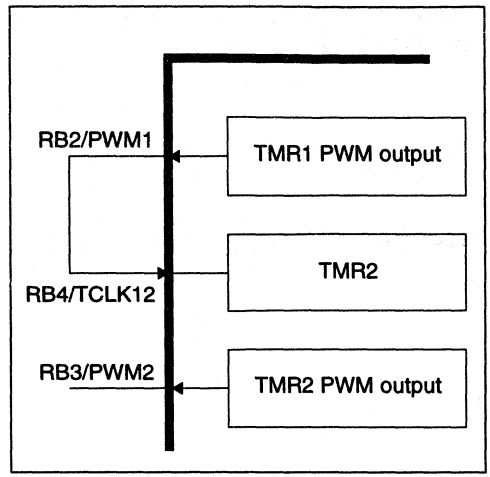
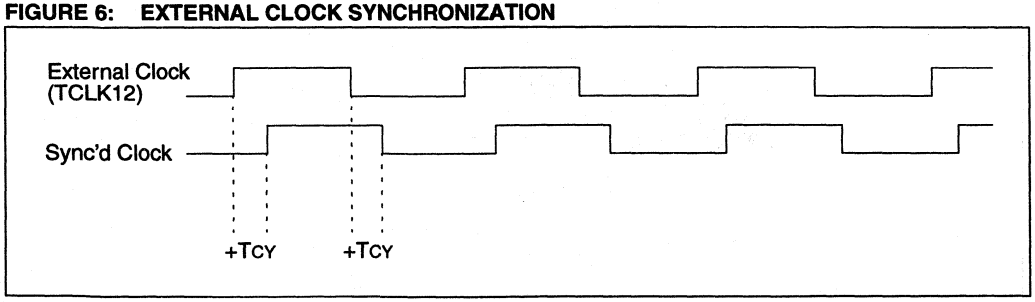


FIGURE 6: EXTERNAL CLOCK SYNCHRONIZATION



© 1997 Microchip Technology Inc.

Another use is where precise timing of updates need to be done, but not at the frequency of the PWM output. In this discussion, TMR1 is used as the time-base of a constant frequency PWM output. TMR1 uses the internal clock of the device and TMR2 uses the external clock input. TMR2 will get the clock input from the PWM2 output.

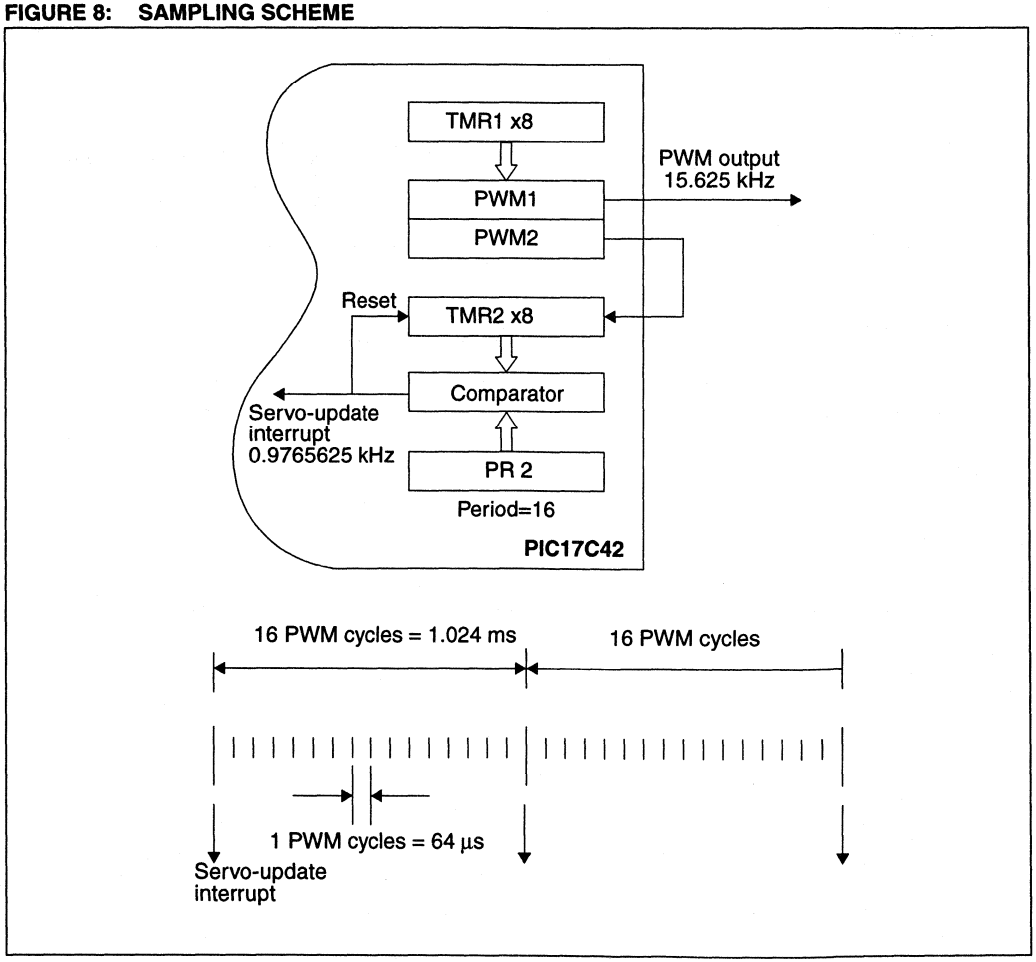
The PWM output is a constant frequency variable duty cycle output. The PW1DCH:PW1DCL register pair contain the variable duty cycle value of PWM1 output. The PW2DCH:PW2DCL register pair is set for a fixed duty cycle (50%) for the PWM2 output.

The PWM outputs could be programmed to have a frequency of 20 kHz, so to reduce audible noise. The PWM2 signal is connected to the RB4/TCLK12, as shown in Figure 8. The PR2 register could be loaded with 14h (20), to give an interrupt every 1 kHz. This interrupt can then trigger tasks, such as updating the duty cycle of PWM1. This is useful in motor control as well as other applications where the update rate is less then the PWM frequency.

CONCLUSION

The PIC17C42's PWM features offer a high performance solution at a lower system cost than previously available. The versatility of PWMs make the PIC17C42 ideal for motor control applications (ses AN532) and many industrial control applications.

FIGURE 8: SAMPLING SCHEME



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: LISTING FILE

```
MPASM 01.40 Released          AN564_P1.ASM   1-16-1997  16:47:52          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001             PROCESSOR               17C42
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;
                     00005 ;       Program:          AN564-P1.ASM
                     00006 ;       Revision Date:
                     00007 ;                         1-15-97  Compatibility with MPASMWIN 1.40
                     00008 ;
                     00009 ;*****************************************************************
                     00010 ;
                     00011
                     00012 ; This is the basic outline for a program that generates a
                     00013 ; variable PWM output. The PWM's period and duty cycle can
                     00014 ; be varied. The new period (NEW_PR1) and the new duty cycle
                     00015 ; (NEW_DC1 and NEW_DC1Q) are loaded by the user program.
                     00016 ; The peripheral interrupt routine loads the new period value
                     00017 ; (frequency) into the PR1 register. A subroutine (PWM_UD)
                     00018 ; is also used to ensure that the 10-bit duty cycle registers
                     00019 ; are updated in the same PWM cycle, i.e. the timer match does not
                     00020 ; occur between two duty cycle register writes.
                     00021 ;
                     00022 ; The duty cycle value gets latched on the overflow (Period match)
                     00023 ; of the timer. The period value gets modified as soon as the period
                     00024 ; register is changed. Therefore care must be taken in updating
                     00025 ; the period register. In cases where the period value is modified
                     00026 ; to a smaller value, we must ensure that the Timer counter is less
                     00027 ; then this value when the period register is updated (TMR1 < new PR1).
                     00028 ; If TMR1 is greater then PR1, the counter will count to FFh, rollover
                     00029 ; to 00H, and only cause the overflow interrupt when it then reaches
                     00030 ; the period value. This would give a wrong PWM output.
                     00031 ;
                     00032 ; In this example the event which cause the PWM to be updated
                     00033 ; is an asynchronous event. A low frequency signal was placed on
                     00034 ; port pin RB0.
                     00035 ; For a high level the PWM registers are updated as follows:
                     00036 ;    PR1 = 7Fh, PW1DCH = 3Fh, and PW1DCL = 40h
                     00037 ; For a low level the PWM registers are updated as follows:
                     00038 ;    PR1 = 1Fh, PW1DCH = 07h, and PW1DCL = 80h
                     00039 ;
                     00040 ; Do the EQUate table
                     00041 ;
00000020             00042 NEW_DC1          EQU  0x20 ; New PWM1 duty cycle value
00000021             00043 NEW_DC1Q         EQU  0x21 ;
00000022             00044 NEW_PR1          EQU  0x22 ; New PWM1 period value
00000025             00045 PWM_WIN          EQU  0x25 ; Register for the PWM window cycle count
00000026             00046 CALC_PR          EQU  0x26 ; Calculated period value
00000027             00047 FLAG_REG         EQU  0x27 ; Register for flag bits
                     00048 ;
0000001A             00049 DC1H             EQU  0x1A ; PWM registers for high time
0000001B             00050 DC1QH            EQU  0x1B
0000001C             00051 PR1H             EQU  0x1C
                     00052 ;
0000001D             00053 DC1L             EQU  0x1D  ; PWM registers for low time
```

```
0000001E    00054 DC1QL             EQU  0x1E
0000001F    00055 PR1L              EQU  0x1F
            00056 ;
            00057 ;
000007FF    00058 END_OF_PROG_MEM   EQU  0x07FF
            00059 ;
00000004    00060 ALUSTA            EQU  0x04
00000006    00061 CPUSTA            EQU  0x06
00000007    00062 INTSTA            EQU  0x07
0000000A    00063 W                 EQU  0x0A
            00064 ;
00000011    00065 DDRB              EQU  0x11  ; Bank 0
00000012    00066 PORTB             EQU  0x12
            00067 ;
00000016    00068 PIR               EQU  0x16  ; Bank 1
00000017    00069 PIE               EQU  0x17
            00070 ;
00000010    00071 TMR1              EQU  0x10  ; Bank 2
00000011    00072 TMR2              EQU  0x11
00000012    00073 TMR31             EQU  0x12
00000013    00074 TMR3h             EQU  0x13
00000014    00075 PR1               EQU  0x14
00000015    00076 PR2               EQU  0x15
00000016    00077 PR3L              EQU  0x16
00000017    00078 PR3h              EQU  0x17
            00079 ;
00000010    00080 PW1DCL            EQU  0x10  ; Bank 3
00000011    00081 PW2DCL            EQU  0x11
00000012    00082 PW1DCH            EQU  0x12
00000013    00083 PW2DCH            EQU  0x13
00000016    00084 TCON1             EQU  0x16
00000017    00085 TCON2             EQU  0x17
            00086  PAGE
0000        00087                   ORG  0x0000   ; Origin for the RESET vector
0000 C02B   00088                   GOTO START    ; On reset, go to the start of
            00089                                 ;   the program
0008        00090                   ORG  0x0008   ; Origin for the external RA0/INT
            00091                                 ;   interrupt vector
0008 C07C   00092                   GOTO EXT_INT  ; Goto the ext. interrupt
            00093                                 ;   on RA0/INT routine
0010        00094                   ORG  0x0010   ; Origin for the TMR0
            00095                                 ;   overflow interrupt vector
0010 C07D   00096                   GOTO TMR0INT  ; Goto the TMR0 overflow interrupt
            00097                                 ;   routine
0018        00098                   ORG  0x0018   ; Origin for the external
            00099                                 ;   RA1/T0CKI interrupt vector
0018 C07E   00100                   GOTO T0_INT   ; Goto the ext. interrupt on
            00101                                 ;   RA1/T0CKI routine
0020        00102                   ORG  0x0020   ; Origin for the interrupt vector
            00103                                 ;   of any enabled peripheral
            00104 ;
            00105 ; The interrupt routine for any peripheral interrupt, This routine
            00106 ; only deals with Timer1 interrupt.
            00107 ;
            00108 ; Time required to execute interrupt routine. Not including
            00109 ; interrupt latency (time to enter into the interrupt routine)
            00110 ;
            00111 ;       case1 - only T1 overflow = 12 cycles
            00112 ;       case2 - Other            = Infinite Loop
            00113 ;
            00114 ;
0020 B801   00115 PER_INT MOVLB   1                     ; Select register Bank 1
0021 9416   00116                 BTFSS   PIR,4         ; Did Timer1 overflow?
0022 C022   00117 ERR1    GOTO    ERR1                  ; Not a Timer1 overflow.
            00118                                       ;   No other interrupts should
            00119                                       ;   be enabled, so error.
```


© 1997 Microchip Technology Inc.

```
                00120 ;
                00121 ; Once the enabled Timer1 overflow occurs, the period register
                00122 ; is loaded. This PWM waveform will remain until the PWM duty
                00123 ; cycle and / or period is updated. Until such update, there is no
                00124 ; S/W overhead from T1 interrupts (T1 interrupts can be disabled).
                00125 ;
                00126 ; NOTE: If PW1DCH >= PR1, then the duty cycle of this PWM output
                00127 ;       is 100%.
                00128 ;
                00129 ; NOTE: The new Period register (PR1) value, must always be greater
                00130 ;       than the value in the Timer1 register (TMR1). If a PR1 value
                00131 ;       is loaded that is less then the TMR1 value, the timer will
                00132 ;       continue to count until it reaches the PR1 value. I.E. TMR1
                00133 ;       will overflow at FFh and the count to the new PR1 value.
                00134 ;       Minimum PR1 value is 0Ah, due to time to load new values and
                00135 ;       execute the peripheral interrupt service routine.
                00136 ;
0023 8C16       00137 T1OVFL      BCF      PIR,4        ; Clear Overflow interrupt flag
0024 B802       00138             MOVLB    2            ; Bank2
0025 7422       00139             MOVFP    NEW_PR1,PR1  ; Load this period value
0026 B801       00140             MOVLB    1            ; Bank 0
0027 8C17       00141             BCF      PIE, 4       ; Disable T1 interrupt
                00142                                   ;   (until transition on PORTB0)
0028 B800       00143             MOVLB    0            ; Bank 0
0029 3F12       00144             BTG      PORTB, 7  ; Transition PortB 7 pin (H->L, or L->H)
002A 0005       00145             RETFIE                ; Return from Interrupt
                00146  PAGE
                00147 ;
                00148 ; This is the start of the program.
                00149 ;
002B 8406       00150 START       BSF      CPUSTA,4     ; Disable ALL interrupts via the
                00151                                   ;   Global Interrupt Disable
                00152                                   ;   (GLINTD) bit.
                00153                                   ;
002C            00154 MAIN                              ; Place Main program here
002C B803       00155             MOVLB    3            ; Select register Bank 3
002D 2817       00156             CLRF     TCON2,0      ; Stop the timers, Single Capture
002E B070       00157             MOVLW    0x070        ; Initalize TCON1 so that
002F 0116       00158             MOVWF    TCON1        ;   T1 (8-bit), T2 (8-bit),
                00159                                   ;   and T3 run off the internal
                00160                                   ;   system clock. Timer3 uses
                00161                                   ;   period register
0030 B00D       00162             MOVLW    0x0D         ; Load the PWM window cycle value
0031 0125       00163             MOVWF    PWM_WIN      ;
                00164 ;
0032 B800       00165             MOVLB    0            ; Select register Bank 0
0033 2B11       00166             SETF     DDRB, 1      ; Port B is an input
0034 2912       00167             CLRF     PORTB, 1     ; Set output values to 0 (for PORTB)
0035 8F11       00168             BCF      DDRB, 7 ; PORTB7 is an output used to trigger a scope
0036 2927       00169             CLRF     FLAG_REG, 1  ; Clear the Flag registers
                00170 ;
                00171 ; Load registers with the PWM values that we will switch between. One
                00172 ; set for the time PORTB0 is high and another set for when low.
                00173 ;
                00174 ; For a high level the PWM registers are updated as follows:
                00175 ;    PR1 = 7Fh, PW1DCH = 3Fh, and PW1DCL = 40h
                00176 ; 16Mhz gives a period of 31.75 us and a duty cycle of 16.625 us
                00177 ; For a low level the PWM registers are updated as follows:
                00178 ;   PR1 = 1Fh, PW1DCH = 07h, and PW1DCL = 80h
                00179 ; At 16Mhz this gives a period of 7.75 us, and a duty cycle of 6.00 us
                00180 ;
0037 B803       00181             MOVLB    3            ; Bank 3
0038 B03F       00182             MOVLW    0x3F         ; The Duty Cycle initial value is
0039 4A1A       00183             MOVPF    W, DC1H      ;   50% of the initial period
003A B040       00184             MOVLW    0x40         ;
003B 4A1B       00185             MOVPF    W, DC1QH     ; Duty Cycle low = 01
```

```
003C B007     00186          MOVLW  0x07          ; The Duty Cycle initial value is
003D 4A1D     00187          MOVPF  W, DC1L       ;   25% of the initial period
003E B080     00188          MOVLW  0x80          ;
003F 4A1E     00189          MOVPF  W, DC1QL      ; Duty Cycle low = 10
              00190 ;
0040 B802     00191          MOVLB  2             ; Bank 2
0041 B07F     00192          MOVLW  0x7F          ;
0042 4A1C     00193          MOVPF  W, PR1H       ; The initial period value is 50%
              00194                               ;   of full scale (for High)
0043 B01F     00195          MOVLW  0x1F
0044 4A1F     00196          MOVPF  W, PR1L       ; The initial period value is 12.5%
              00197                               ;   of full scale (for Low)
              00198 ;
              00199 ;
              00200 ; Default PWM values should be set, and the timer should be started
              00201 ; and the interrupts enabled.
              00202 ;
0045 B0F0     00203          MOVLW  0xF0          ; Load the Period register
0046 0114     00204          MOVWF  PR1           ;
0047 B803     00205          MOVLB  3             ; Select register Bank 3
0048 B0C0     00206          MOVLW  0xC0          ; Load the T1 duty cycle register
0049 0112     00207          MOVWF  PW1DCH        ;
004A 0110     00208          MOVWF  PW1DCL        ; effectively loaded with 0
004B B031     00209          MOVLW  0x31          ;** Enable PWM1 and PWM2 outputs
004C 0117     00210          MOVWF  TCON2         ;**   and turn on Timer1.
              00211
004D 8307     00212          BSF    INTSTA,3      ; Turn on Peripheral Interrupts
004E B801     00213          MOVLB  1             ; Select register Bank 1
004F B010     00214          MOVLW  0x10          ; Enable Timer1 overflow
0050 0117     00215          MOVWF  PI            ;   Interrupts (when GLINTD = 0)
0051 8C06     00216          BCF    CPUSTA,4      ; Enable ALL interrupts
0052 B800     00217          MOVLB  0             ; Bank 0
              00218  PAGE
              00219 ;
              00220 ; Only need to update PWM values on the first occurance of a new level
              00221 ; on RB0, Else loop waiting for level to change.
              00222 ;
0053 8827     00223 HIGH1ST BCF     FLAG_REG, 0 ; First time in loop (this cycle)= True
0054 9012     00224 HIGHCYC BTFSS   PORTB, 0    ; Is PortB0 low
0055 C05F     00225         GOTO    LOW1ST      ; PORTB0 = L
0056 9827     00226         BTFSC   FLAG_REG, 0 ; Is this the First High time (this cycle)?
0057 C054     00227         GOTO    HIGHCYC     ; Loop looking for low signal on PortB0
0058 8027     00228         BSF     FLAG_REG, 0 ; Set First time in loop (this cycle)=False
              00229 ;
              00230 ; Here is where we update the PWM values (period and Duty cycle)
              00231 ; for high level.
0059 B803     00232         MOVLB   3               ; Bank 3
005A 5A20     00233         MOVPF   DC1H, NEW_DC1   ;
005B 5B21     00234         MOVPF   DC1QH, NEW_DC1Q ;
005C 5C22     00235         MOVPF   PR1H, NEW_PR1   ;
005D E06B     00236         CALL    PWM1_UD         ;
005E C054     00237         GOTO    HIGHCYC         ; Loop looking for low signal on PortB0
              00238 ;
              00239 ;
005F 8827     00240 LOW1ST   BCF    FLAG_REG, 0    ; First time in loop (this cycle)=True
0060 9812     00241 LOWCYC   BTFSC  PORTB, 0       ; Is PortB0 high
0061 C053     00242          GOTO   HIGH1ST        ; PORTB0 = H
0062 9827     00243          BTFSC  FLAG_REG, 0  ; Is this the First Low time (this cycle)?
0063 C060     00244          GOTO   LOWCYC         ; Loop looking for high signal on PortB0
0064 8027     00245          BSF    FLAG_REG, 0  ; First time in loop (this cycle) = False
              00246 ;
              00247 ; Here is where we update the PWM values (period & Duty cycle) for low level.
              00248 ;
0065 B803     00249          MOVLB  3
0066 5D20     00250          MOVPF  DC1L, NEW_DC1   ;
0067 5E21     00251          MOVPF  DC1QL, NEW_DC1Q ;
```


© 1997 Microchip Technology Inc.

```
0068 5F22     00252          MOVPF  PR1L, NEW_PR1   ;
0069 E06B     00253          CALL   PWM1_UD        ;
006A C060     00254          GOTO   LOWCYC         ; Loop looking for high signal on PortB0
              00255
              00256  PAGE
              00257 ;
              00258 ; This code segment ensure that all PWM values (period and duty cycle)
              00259 ; are updated at the same time. This is done by ensuring that the Timer
              00260 ; is at least PWM_WIN (0Dh) cycles before the PR1 value
              00261 ; (PR1 - PWM_WIN > TMR1).If not a "glitch" could occur in the PWM wave
              00262 ; form. When only the 1st duty cycle register is latched for this PWM
              00263 ; cycle, and the following PWM periodwill latch the 2nd duty cycle
              00264 ; register.
006B 8406     00265 PWM1_UD BSF CPUSTA, 4          ; Disable Global Interrupts
006C B802     00266             MOVLB  2           ; Bank 2
006D 6A10     00267             MOVFP  TMR1, W     ; Load W reg. with Timer1 value
006E 0414     00268             SUBWF  PR1, 0      ; PR1 - TMR1 -> W reg.
006F 3025     00269             CPFSLT PWM_WIN     ; Check if Timer1 is about to overflow
0070 C06B     00270             GOTO   PWM1_UD     ; Overflow would have occurred during
              00271                                ;   PWM updates, Delay a few cycles
0071 B803     00272             MOVLB  3           ; Bank 3
0072 6A20     00273             MOVFP  NEW_DC1, W  ; Your New PWM MSB
0073 0112     00274             MOVWF  PW1DCH      ; Loaded in duty cycle buffer
0074 6A21     00275             MOVFP  NEW_DC1Q, W; Your New PWM LSB
0075 0110     00276             MOVWF  PW1DCL      ; Loaded in duty cycle buffer
0076 B801     00277             MOVLB  1           ; Back to Bank 1
0077 8C16     00278             BCF    PIR, 4      ; Clear T1 Overflow interrupt flag
0078 8417     00279             BSF    PIE, 4      ; Enable T1 int
0079 8C06     00280             BCF    CPUSTA, 4   ; Enable Global Interrupts
007A B800     00281             MOVLB  0           ; Bank 0
007B 0002     00282             RETURN             ;** this does not need to be implemented
              00283                                ;**  as a subroutine.
              00284 ;
              00285 ; Other Interrupt routines. (Not utilized in this example)
              00286 ;
007C 0005     00287 EXT_INT RETFIE                 ; RA0/INT interrupt routine
              00288                                ;   (NOT used in this program)
007D 0005     00289 TMR0INT RETFIE                 ; TMR0 overflow interrupt routine
              00290                                ;   (NOT used in this program)
007E 0005     00291 T0_INT  RETFIE                 ; RA1/T0CKI interrupt routine
              00292                                ;   (NOT used in this program)
              00293                                ;
007F C02B     00294 SRESET  GOTO START             ; If program became lost, goto
              00295                                ;   START and reinitalize.
              00296 ;
              00297 ;
              00298 ; When the executed address is NOT in the program range, the
              00299 ; 16-bit address should contain all 1's (a CALL 0x1FFF). At
              00300 ; this location you could branch to a routine to recover or
              00301 ; shut down from the invalid program execution.
              00302 ;
07FF          00303            ORG    END_OF_PROG_MEM ;
07FF C07F     00304            GOTO   SRESET        ; The program has lost it's mind,
              00305                                ;   do a system reset
              00306                 END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)


0000 : X-------X------- X-------X------- XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
07C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   101
```

```
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN583

Implementation of the Data Encryption Standard Using PIC17C42

Authors: *Al Lovrich*
Mark Palmer
Microchip Technology Inc.

INTRODUCTION

In January 1977, The United States government adopted a product cipher developed by IBM as its official encryption standard [1]. This algorithm, called the Data Encryption Standard (DES), has been adopted as a worldwide standard for data encryption by ISO (International Standards Organization) [2, 3]. This application note describes the implementation of the DES algorithm on PIC17C42.

THE DATA ENCRYPTION STANDARD

The DES algorithm is a substitution cipher which takes a block of 64 bits of input (plaintext) into a unique block of 64 bits of output (ciphertext), under the control of a 64-bit key, which is known only to the people intended to read the message. In this system, plaintext information is divided into several blocks which are then operated upon independently to generate a sequence of ciphertext blocks. The basic idea behind DES is to build a strong system out of simple, individually weak, components. The DES encryption system is based on a system of transpositions and permutations. The permutation box or P-box, is used to transpose, or map a sequence of input values to another sequence of values of the same length. Substitutions are performed by what are called S-boxes. A combination of the S-boxes and P-boxes can be viewed as a decoder/coder operation, where the output is simply a linear mapping of the input values. Each combination of the S-box and P-box comprises a single weak component of the algorithm. By including a sufficiently large number of stages in the product cipher, the output can be made to be a nonlinear function of the input.

The mapping of input to output is one-to-one and invertible, since the encrypted messages can be decrypted. The DES has three distinct components: key schedule, cipher function, and invertibility.

KEY SCHEDULE

The DES uses a 64-bit key for encryption and decryption process. Initially, the original 64-bit key is reduced to 56-bit by ignoring every eighth bit. In general these bits are used as parity bits to make sure that there were no errors when entering the key or during key transmission. After the 56-bit key is extracted a different 48-bit key, referred to as subkey, is generated for each of the 16 rounds of the DES. These keys, K_i, are determined as shown in Figure 1. The 56-bit key is divided into two 28-bit halves C_i and D_i which are then shifted left by either 1 or 2 digits, depending on the round. Table 1 shows the number of circular left shifts for C_i and D_i halves. After the shifting operation, 48 out of the 56 bits are selected. Since this operation permutes the order of the bits as well as selecting a subset of the original bits, it is called compression permutation or permuted choice. The permuted choice 1 and permuted choice 2 matrices are shown in Figure 2 and Figure 3 respectively.

TABLE 1: LEFT SHIFTS FOR KEY GENERATION

Iteration	# of left shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

4

© 1997 Microchip Technology Inc.

FIGURE 1: KEY GENERATION

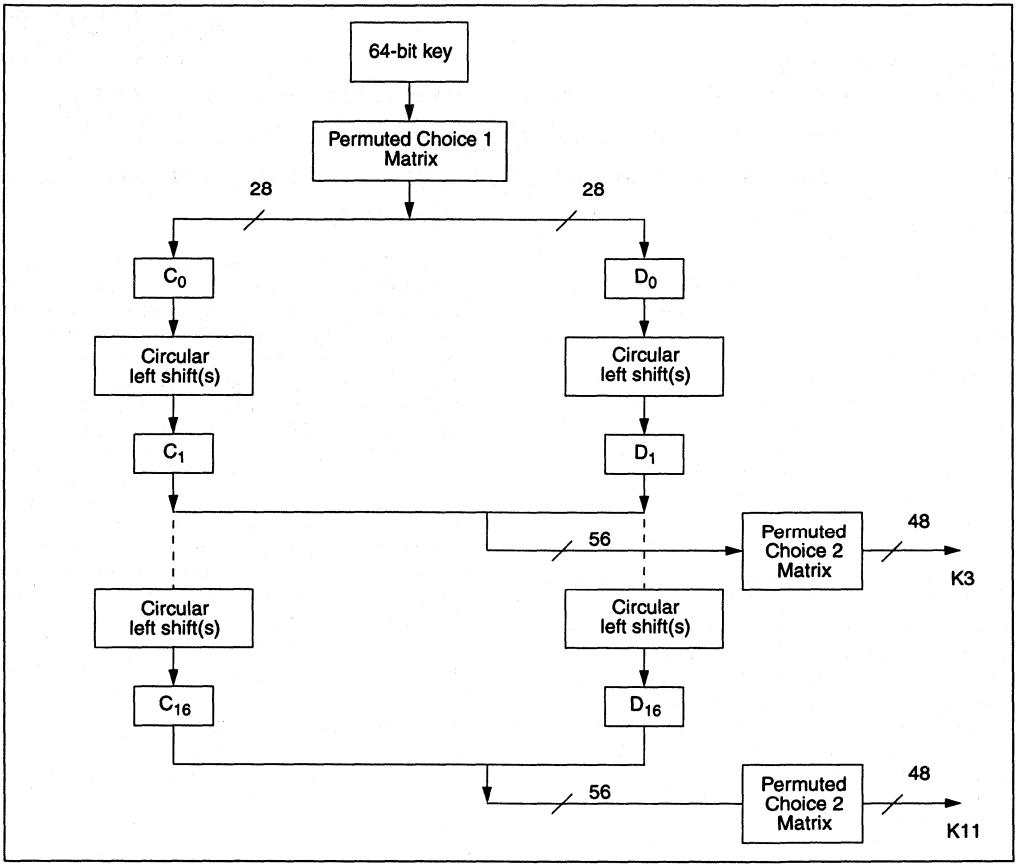


FIGURE 2: PERMUTED CHOICE 1 MATRIX

57	49	41	33	25	17	9	1
58	50	42	34	26	18	10	2
59	51	43	35	27	19	11	3
60	52	44	36	63	55	47	39
31	23	15	7	62	54	46	38
30	22	14	6	61	53	45	37

FIGURE 3: PERMUTED CHOICE 2 MATRIX

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

© 1997 Microchip Technology Inc.

CIPHER FUNCTION

The strength of the DES is based on the cipher function component. This is a fixed, highly nonlinear function which guarantees that each bit of the ciphertext depend on every bit of the plaintext and every bit of the key.

After an initial permutation, the 64-bit block of plaintext is broken into a right half and left half, each 32 bits long. This step is followed by 16 identical rounds of operation, called function *f*, that combines the data with a 48-bit key, K_i. At each stage *i*, the inputs are the left block L_{i-1} and the right block R_{i-1} of the previous stage, and the outputs are the left shift block L_i and right block R_i of this stage. The outputs of L_i and R_i of each stage are computed from L_{i-1} and R_{i-1}, and a subkey K_i that is generated from the encryption key. In other words a round of the DES can be shown as:

$$L_i = R_i$$

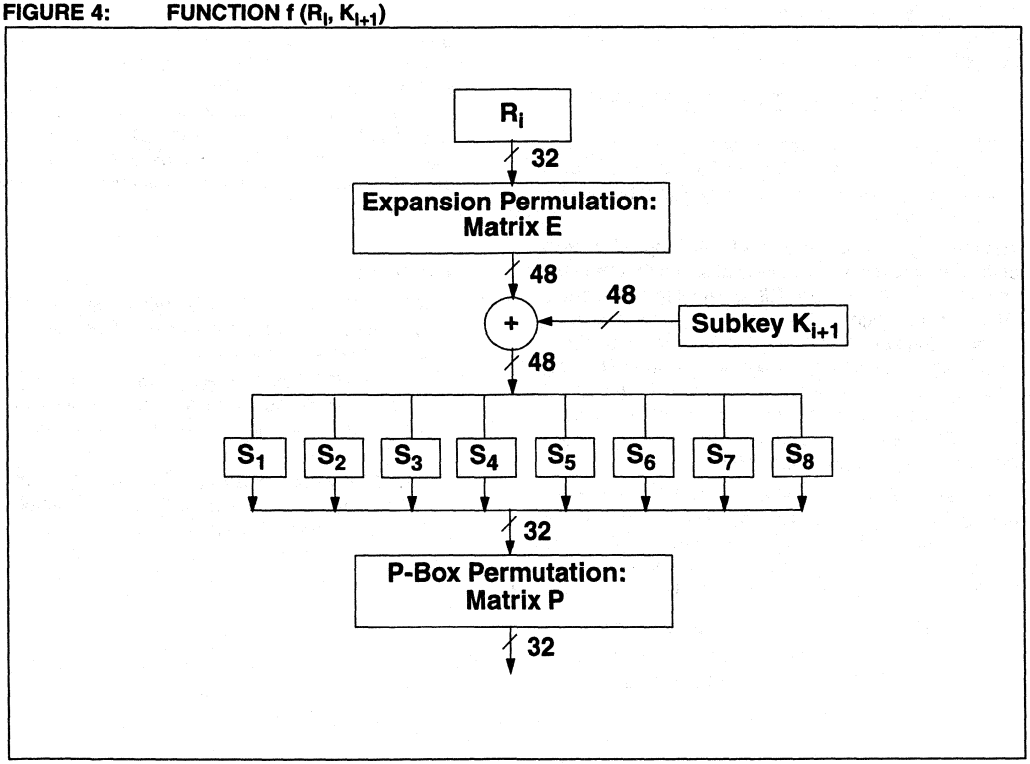
$$R_i = L_{i-1} \; XOR \; f(R_{i-1}, K_i)$$

All the complexity of the DES algorithm lies in the function f, as shown in Figure 4. The function has four steps that are carried out in sequence. First a 48-bit number, *E*, is constructed by expanding the 32-bit previous right value, R_{i-1}, according to a fixed transposition and duplication rule. Then, K_i and R_{i-1} are XORed together, generating a 48-bit result. This output is then partitioned into eight groups of 6-bits each, each of which is fed into a different S-box or substitution box. The S-boxes generate four instead of six outputs. In other words, the 64-bit input is mapped into a 32-bit output. Each S-box is a table of 4 rows and 16 columns. Each entry in the box is a 4-bit number. The six input bits of the S-box specify under which row and column number to look for the output. Figure 7 shows the 8 S-boxes.

The six input bits specify an entry in the Sbox in a particular fashion as follows: the first and last bits of the sequence, taken together, represent a number between 0 and 3 (row entry), while the middle 4-bits represent a number between 0 and 15 (column entry). The output is simply the entry that corresponds to the (row, column) entry. For example, that the input to the first S-box is the binary sequence `110010`. The first and last bits combine to form 10 which corresponds to the third row of the S-box. The middle 4-bits are combined to form `1001` which corresponds to the ninth column of the for `110010`. The substitution boxes are the most critical step in the DES algorithm and more than anything else give DES its security.

4

FIGURE 4: FUNCTION f (R_i, K_{i+1})



© 1997 Microchip Technology Inc.

Finally, the last stage consists of a permutation stage that generates a 32-bit output. After the 16 rounds, the left and right halves are joined, and a final permutation generates the ciphertext. The final permutation is the inverse of the initial permutation. Figure 10 shows a block diagram of the enciphering portion of the algorithm. The reverse process of deciphering is shown in Figure 11. The initial permutation and inverse initial permutation matrices in Figure 10 are shown in Figure 8 and Figure 9. Where the algorithm requires bit manipulation of a stream of data according to a matrix, the matrix is read from left to right, top to bottom, and interpreted as the bit position in the output block. For example, the initial permutation matrix transposes bit1 to bit58, bit2 to bit50, bit3 to bit42, etc.

INVERTIBILITY

The DES cipher function is not necessarily invertible, meaning to decode a message, it is not necessary to recover the input to the cipher function from its output and a knowledge of the key. In fact the cipher function must be highly nonlinear to be resistant to plaintext attack (a method used for breaking a given algorithm). Invertibility of the DES is that one half-word of the output is precisely the bit configuration which was used to encode the other half, with the aid of the particular stage subkey. Therefore, by using the subkeys in reverse order, the encryption process can be reversed. This is really the reason that one half-word is always passed through unchanged - to provide the means of decrypting the other half-word.

PIC17C42 IMPLEMENTATION OF DES

CPU processing is required to generate the encryption key into the DES subkeys. The 64-bit encryption key is reduced to 56-bits, by ignoring every eighth bit, usually used as parity bit.

The majority of the DES code is for the Implementation of the permutation of the block of bits. The 56-bits of the key, stored in K1 through K8, scrambled-bit output is stored in the eight bytes D0-D7. The scrambling is accomplished by constructing D0-D7, one bit at a time. This is accomplished by initializing the D0 to D7 locations to a known state (cleared). Then the 64-bits of plain text are processed through the Initial Permutation Matrix (IP), which permutes the plain text and divides the information into two 32-bit blocks.

The use of Indirect addressing and the PICmicro™ single word instructions allows tight efficient coding of the DES algorithm. These bit testing capabilities allows the same code structure to generate the different subkey blocks. This permutation macro looks like:

```
Permute   Macro     KEY,TEST,BIT
          BTFS C    KEY,TEST,BIT
          BSF       INDOF0,BIT
          endm
```

Where KEY is the DES key and TEST is the bit is the KEY being tested. If the KEY<TEST> bit is set, then the bit position (BIT) in the data location pointed to by INDF0 is set.

The main algorithm requires that the 16 subkeys, each 48-bits long, be generated. These 16 subkeys are then used at the 16 stages of the algorithm.

Using the generated subkeys, the incoming stream of bits can be encrypted or decrypted. Table 2 shows the requirements of the DES algorithm.

TABLE 2: DES ALGORITHM REQUIREMENTS

		Execution time	
Function	Program Memory words	Instruction cycles	ms
Key management and subkey generation	382	2729	0.436
Encryption	789	7714	1.234

A bit rate of about 51 Kbps baud can be achieved, with a device utilization of 100%. This makes the PIC17C42 a price/performance leader for DES algorithms.

CONCLUSION

We have demonstrated the implementation of the DES algorithm on the PIC17C42 microcontroller. The 160 ns cycle time of the PIC17C42 makes possible a half-duplex rate of 51 Kbps for the DES. This rate is as good or superior to other implementations of the algorithm. The high performance of the PIC17C42 provides a low-cost alternative to many dedicated solutions resulting in minimum system cost because of the programmability of the device.

References

1. NBS FIPS PUB 46, "Data Encryption Standard," National Bureau of Standards, US Department of Commerce, January 1977.
2. SO DIS 8730, "Banking Requirements for Message Authentication (Wholesale)," Association for Payment Clearing Services, London, July 1987.
3. ISO DIS 8732, "Banking Key Management (Wholesale)," Association for Payment Clearing Services, London, December 1987.

© 1997 Microchip Technology Inc.

FIGURE 5: MATRIX E

32	1	2	3	4	5	4	5
6	7	8	9	8	9	10	11
12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21
22	23	24	25	24	25	26	27
28	29	28	29	30	31	32	1

FIGURE 6: MATRIX P

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

FIGURE 7: S MATRICES

```
/* S1 */
14 4 13 1 2 15 11 8 3 10 6 12 5 9 0 7
0 15 7 4 14 2 13 1 10 6 12 11 9 5 3 8
4 1 14 8 13 6 2 11 15 12 9 7 3 10 5 0
15 12 8 2 4 9 1 7 5 11 3 14 10 0 6 13

/* S2 */
15 1 8 14 6 11 3 4 9 7 2 13 12 0 5 10
3 13 4 7 15 2 8 14 12 0 1 10 6 9 11 5
0 14 7 11 10 4 13 1 5 8 12 6 9 3 2 15
13 8 10 1 3 15 4 2 11 6 7 12 0 5 14 9

/* S3 */
10 0 9 14 6 3 15 5 1 13 12 7 11 4 2 8
13 7 0 9 3 4 6 10 2 8 5 14 12 11 15 1
13 6 4 9 8 15 3 0 11 1 2 12 5 10 14 7
1 10 13 0 6 9 8 7 4 15 14 3 11 5 2 12

/* S4 */
7 13 14 3 0 6 9 10 1 2 8 5 11 12 4 15
13 8 11 5 6 15 0 3 4 7 2 12 1 10 14 9
10 6 9 0 12 11 7 13 15 1 3 14 5 2 8 4
3 15 0 6 10 1 13 8 9 4 5 11 12 7 2 14

/* S5 */
2 12 4 1 7 10 11 6 8 5 3 15 13 0 14 9
14 11 2 12 4 7 13 1 5 0 15 10 3 9 8 6
4 2 1 11 10 13 7 8 15 9 12 5 6 3 0 14
11 8 12 7 1 14 2 13 6 15 0 9 10 4 5 3
```

FIGURE 7: S matrices (CONT.)

```
/* S6 */
12 1 10 15 9 2 6 8 0 13 3 4 14 7 5 11
10 15 4 2 7 12 9 5 6 1 13 14 0 11 3 8
9 14 15 5 2 8 12 3 7 0 4 10 1 13 11 6
4 3 2 12 9 5 15 10 11 14 1 7 6 0 8 13

/* S7 */
4 11 2 14 15 0 8 13 3 12 9 7 5 10 6 1
13 0 11 7 4 9 1 10 14 3 5 12 2 15 8 6
1 4 11 13 12 3 7 14 10 15 6 8 0 5 9 2
6 11 13 8 1 4 10 7 9 5 0 15 14 2 3 12

/* S8 */
13 2 8 4 6 15 11 1 10 9 3 14 5 0 12 7
1 15 13 8 10 3 7 4 12 5 6 11 0 14 9 2
7 11 4 1 9 12 14 2 0 6 10 13 15 3 5 8
2 1 14 7 4 10 8 13 15 12 9 0 3 5 6 11
```

FIGURE 8: INITIAL PERMUTATION MATRIX

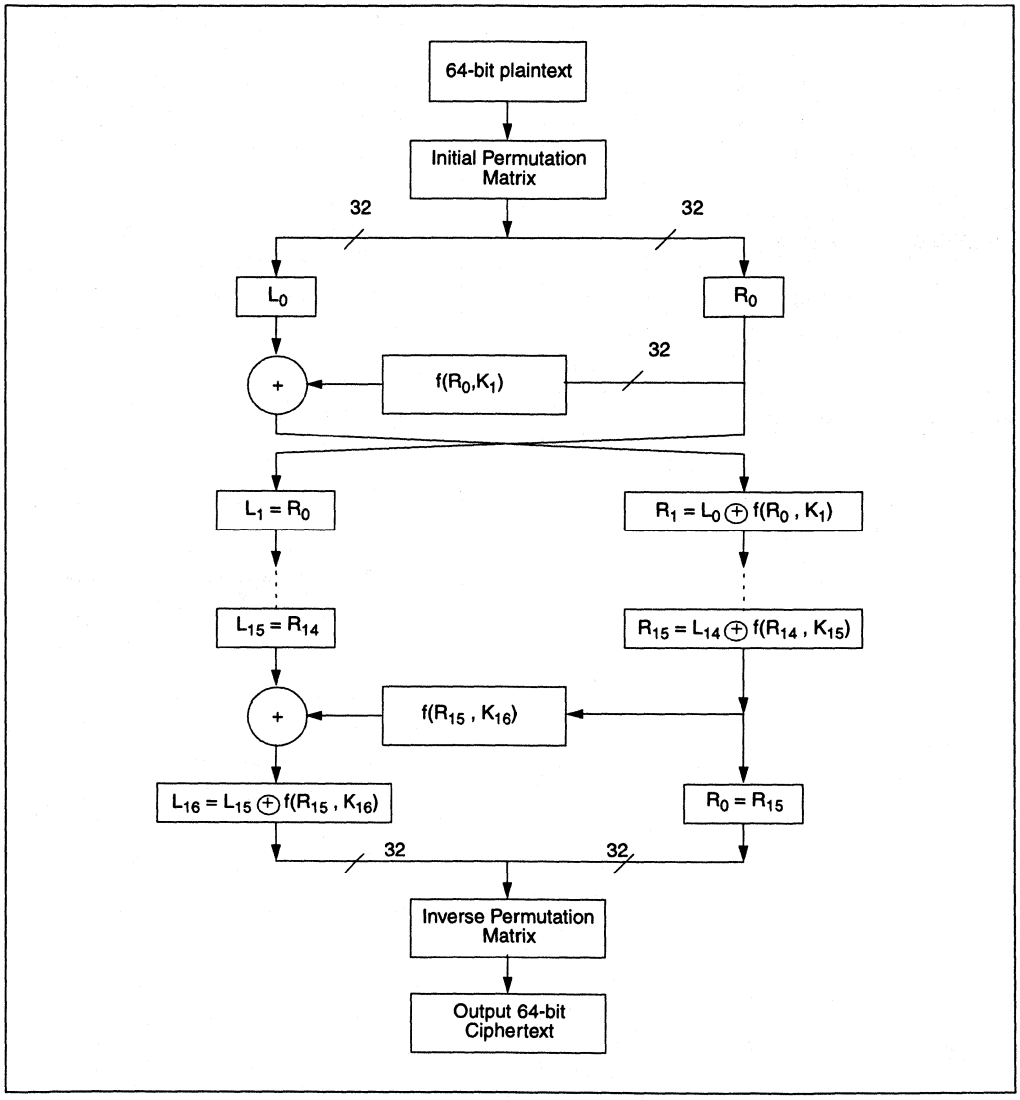
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

FIGURE 9: INVERSE PERMUTATION MATRIX

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	14	52	20	60	28
35	3	43	13	51	19	59	27
34	2	42	12	50	18	58	26
33	1	41	11	49	17	57	25

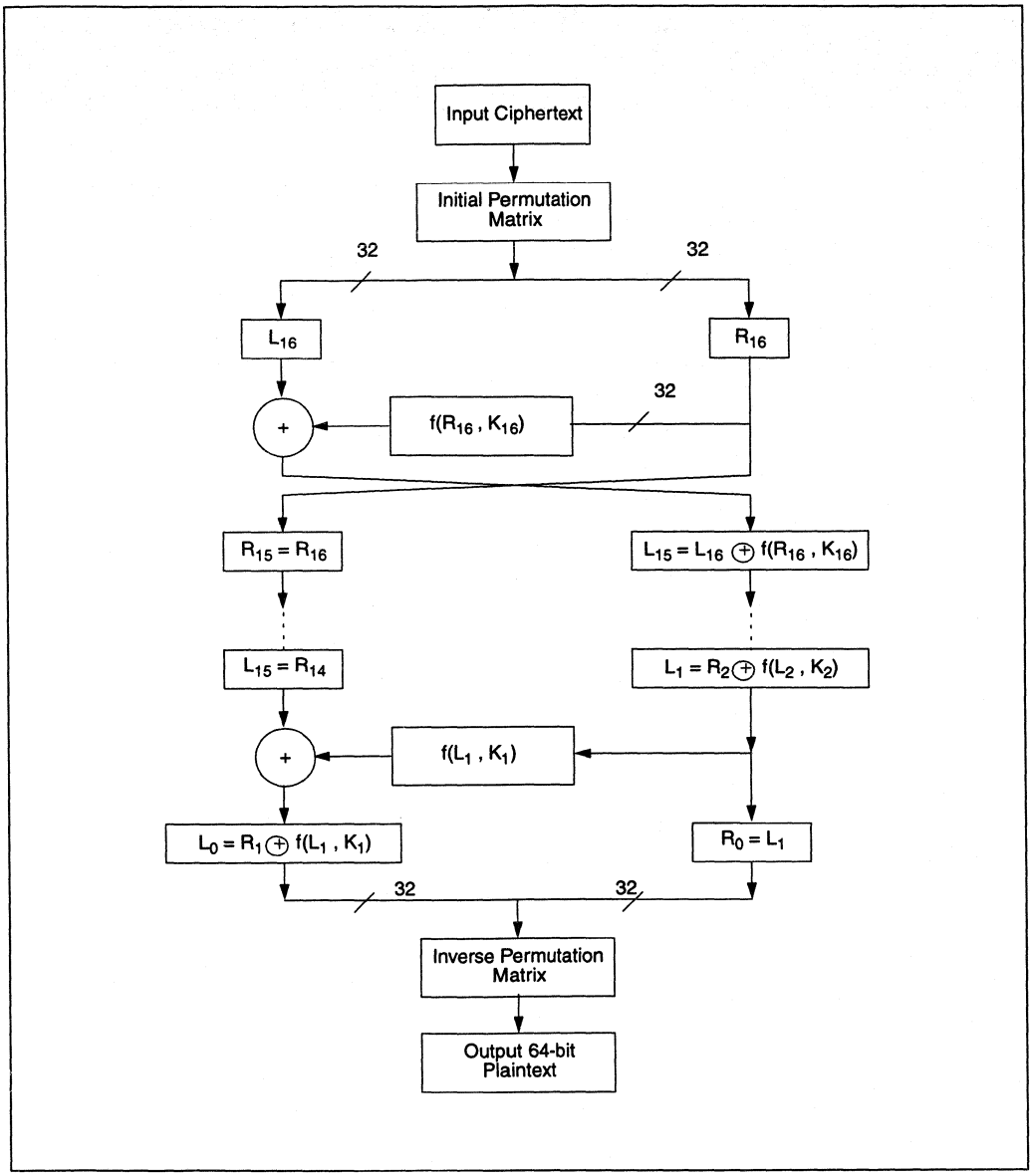
© 1997 Microchip Technology Inc.

FIGURE 10: DES ENCRYPTION BLOCK DIAGRAM



© 1997 Microchip Technology Inc.

FIGURE 11: DES DECRYPTION BLOCK DIAGRAM



APPENDIX A:

Note: The PIC17C42 code implementing the DES is not published because it falls within the U. S. Department of State Export control Regulations.

Please contact your local Microchip sales office to obtain a copy of the code.

© 1997 Microchip Technology Inc.

SECTION 5
PICmicro™ MICROCONTROLLER APPLICATION NOTES

PIC16C5X / PIC16CXXX Math Utility Routines - AN526 5-1
Implementing a Table Read - AN556 5-95
Techniques to Disable Global Interrupts - AN576 5-99
A Real-Time Operating System for PICmicro™ Microcontrollers - AN585 5-105
PICmicro™ Microcontroller Oscillator Design Guide - AN588 5-143
Low Power Design Using PICmicro™ Microcontrollers - AN606 5-161
Power-up Trouble Shooting - AN607 5-177
Clock Design Using Low Power/Cost Techniques - AN615 5-199

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

PIC16C5X / PIC16CXXX Math Utility Routines

Author: *Amar Palacherla*
Microchip Technology Inc.

PLEASE NOTE: *This application note uses the old Microchip Math Routine format. It is intended for reference purposes only and is being provided for those of you still implementing Binary Coded Decimal(BCD) routines. For any new designs, please refer to application notes contained in Microchip's Embedded Control Handbook Volume II - Math Library.*

INTRODUCTION

This application note provides some utility math routines for Microchip's PIC16C5X and PIC16CXXX series of 8-bit microcontrollers. The following math outlines are provided:

- 8x8 unsigned multiply
- 16x16 double precision multiply
- Fixed Point Division (Table 3)
- 16x16 double precision addition
- 16x16 double precision subtraction
- BCD (Binary Coded Decimal) to binary conversion routines
- Binary to BCD conversion routines
- BCD addition
- BCD subtraction
- Square root

These are written in native assembly language and the listing files are provided. They are also available on a disk (MS-DOS®). All the routines provided can be called as subroutines. Most of the routines have two different versions: one optimized for speed and the other optimized for code size. The calling sequence of each routine is explained at the beginning of each listing file.

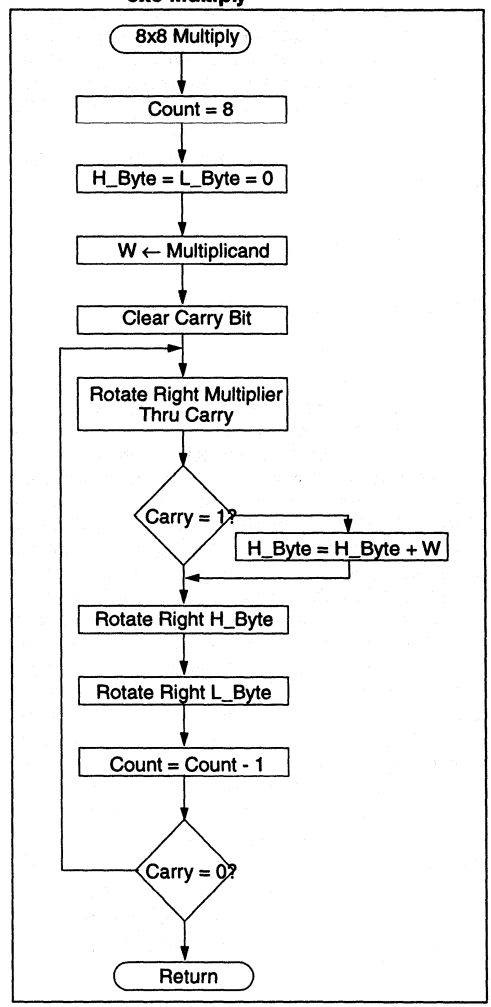
SINGLE PRECISION UNSIGNED MULTIPLICATION (8x8)

This routine computes the product of two 8-bit unsigned numbers and produces a 16-bit result. Two routines are provided: one routine is optimized for speed (by writing a straight line code) and the other routine has been written to reduce the code size (a looped code). The listing of these routines are given in Appendices A and B. The performance specs for the routines are shown in Table 1.

TABLE 1: PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Speed Efficient	35	37
Code Efficient	16	71

FIGURE 1: Flowchart for Unsigned 8x8 Multiply



MS-DOS is a registered trademark of Microsoft Corporation.

© 1997 Microchip Technology Inc.

DOUBLE PRECISION MULTIPLY

This routine computes the product of two 16-bit numbers and produces a 32-bit result. Both signed and unsigned arithmetic are supported. Two routines are provided: one routine is optimized for speed (by writing a straight line code) the other routine has been written to reduce code size (a looped code). The listing of these routines are given in Appendices C and D. The performance specs for routines are shown in Table 2.

TABLE 2: PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Speed Efficient	240	233
Code Efficient	33	333

The code in Appendices C and D has been setup for unsigned arithmetic and the performance specs in the table above is for unsigned arithmetic. If signed arithmetic is desired, edit the line with "`SIGNED equ FALSE`" to "`SIGNED equ TRUE`" then re-assemble the code.

In case of signed multiply, both operands are assumed to be 16-bit 2's complement numbers.

Conditional assembly is supported by MPASM. If you have an older version, please contact the Microchip Technology sales office nearest you.

DOUBLE PRECISION DIVISION

Fixed Point Divide Routines

Fixed point division is fundamentally a conditional shift and subtract operation, resulting in a quotient and a remainder, with standard methods related to simple binary long division by hand calculation. Typically, a processor with n-bit operands uses a fixed accumulator of 2n bits containing the dividend. In standard restoring division, the dividend is left shifted by one bit and the divisor is subtracted from the high half of the accumulator, referred to as the partial remainder. If the result is positive, the divisor was less than or equal to the partial remainder and the corresponding quotient bit in the LSb of the accumulator is set to one. If the result is negative, the divisor was greater than the partial remainder and the dividend is restored by adding back the divisor to the high half of the accumulator and setting the LSb to zero. This process is repeated for each of the n bits in the divisor, resulting in an n-bit quotient in the low half of the accumulator and the n-bit remainder in the high half, and requiring n subtractions and on average n/2 additions [1].

Nonrestoring division, requiring a total of at most n+1 subtractions and additions, offers potential for speed improvement by allowing a negative partial remainder during the calculation with a final addition of the divisor if the final remainder is negative. After the first left shift of the dividend, the divisor is subtracted and the corresponding quotient bit as well as the next add or subtract operation is determined by the carry bit [1].

Unfortunately, no simple method exists for performing two's complement binary division, thereby requiring negate operations during a preprocessing phase. It is important to note that with the dividend initially loaded into the accumulator, an overflow of the final quotient will result if the high half of the dividend is greater than or equal to the divisor [1], indicating that the n-bit range of the quotient will be exceeded.

Because of the inherent byte structure of the PICmicro™ family of microcontrollers, a more creative and efficient implementation of the above algorithms is possible. In what follows, partial remainder is initialized at zero and is separated from the dividend, thereby avoiding any alignment logic overhead and yielding a quotient with the same number of bits as the dividend and a remainder with the same number as the divisor. Furthermore, routines are named in format FXDxxyyz, where xx is the number of bits in the dividend, yy is the number of bits in the divisor, and z indicates a signed or unsigned routine. Macros are used for core sections of each routine, thereby permitting simple switching between restoring and nonrestoring methods. The signed macros are exclusively a variation of the nonrestoring method, taking advantage of the zero MSb of the arguments after the preprocessing negation. Both restoring and nonrestoring macros are included for the unsigned case, with selection based on best worst case or best average performance as desired. For example, the unsigned macros exhibit the following performance data:

		# of Cycles (TCY)		
		32/16	16/16	16/8
restore	max.	561	240	193
	ave.	481	208	173
nonrestore	max.	481	240	190
	ave.	466	233	183

This demonstrates that while the nonrestoring algorithm is preferred for the 32/16 case, the restoring method is preferred for the 16/16 case, with the choice for the 16/8 case a function of user requirements. These optimization complications are a result of trade-offs between the number of instructions required for the restore operations verses the added logic requirements. Finally, additional routines with tacit MSb equal to zero in each argument are included, yielding significant speed improvement. These routines can also be called in the signed case when the arguments are known to be positive for a small benefit.

© 1997 Microchip Technology Inc.

Routines

It is useful to note that the additional routines FXD3115U, FXD1515U, and FXD1507U can be called in a signed divide application in the special case where AARG > 0 and BARG > 0, thereby offering some improvement in performance.

Data RAM Requirements

The following contiguous data RAM locations are used by the fixed point divide routines:

ACC+B0	=	AARG+B0	AARG and ACC
ACC+B1	=	AARG+B1	
ACC+B2	=	AARG+B2	
ACC+B3	=	AARG+B3	
ACC+B4	=	REM+B0	remainder
ACC+B5	=	REM+B1	
SIGN			sign in MSb
BARG+B0			BARG
BARG+B1			
TEMP+B0			temporary storage
TEMP+B1			

where Bx = x.

References

1. Cavanagh, J.J.F., "Digital Computer Arithmetic," McGraw-Hill,1984.
2. Hwang, K., "Computer Arithmetic," John Wiley & Sons, 1979.
3. Scott, N.R., "Computer Number Systems & Arithmetic," Prentice Hall, 1985.

5

TABLE 3: Fixed Point Divide Routines

Routine	Cycles	Function
FXD3216S	414	32-bit/16-bit -> 32.16 signed fixed point divide
FXD3216U	485	32-bit/16-bit -> 32.16 unsigned fixed point divide
FXD3215U	390	32-bit/15-bit -> 32.15 unsigned fixed point divide
FXD3115U	383	31-bit/15-bit -> 31.15 unsigned fixed point divide
FXD1616S	214	16-bit/16-bit -> 16.16 signed fixed point divide
FXD1616U	244	16-bit/16-bit -> 16.16 unsigned fixed point divide
FXD1615U	197	16-bit/15-bit -> 16.15 unsigned fixed point divide
FXD1515U	191	15-bit/15-bit -> 15.15 unsigned fixed point divide
FXD1608S	146	16-bit/08-bit -> 16.08 signed fixed point divide
FXD1608U	196	16-bit/08-bit -> 16.08 unsigned fixed point divide
FXD1607U	130	16-bit/07-bit -> 16.07 unsigned fixed point divide
FXD1507U	125	15-bit/07-bit -> 15.07 unsigned fixed point divide

© 1997 Microchip Technology Inc.

TABLE 4: PIC16CXXX Fixed Point Divide Performance Data

Routine	Max. Cycles	Min. Cycles	Program Memory	Data Memory
16 / 8 Signed	146	135	146	5
16 / 8 Unsigned	196	156	195	4
16 / 7 Unsigned	130	130	129	4
15 / 7 Unsigned	125	125	124	4
16 / 16 Unsigned	214	187	241	7
16 / 16 Unsigned	244	180	243	6
16 / 15 Unsigned	197	182	216	6
16 / 15 Unsigned	191	177	218	6
32 / 16 Unsigned	414	363	476	9
32 / 16 Unsigned	485	459	608	9
32 / 15 Unsigned	390	359	451	8
31 / 15 Unsigned	383	353	442	8

DOUBLE PRECISION ADDITION & SUBTRACTION

This routine adds or subtracts two 16-bit numbers and produces a 16-bit result. This routine is used by other double precision routines. The listing of these routines is given in Appendix E. The performance specs for the routines are shown below:

TABLE 5: PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Addition	7	8
Subtraction	14	17

BCD TO BINARY CONVERSION

This routine converts a five digit BCD number to a 16-bit binary number. The listing of this routine is given in Appendix F. The performance spec for the routine is shown below:

TABLE 6: PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
BCD to Binary	30	121

© 1997 Microchip Technology Inc.

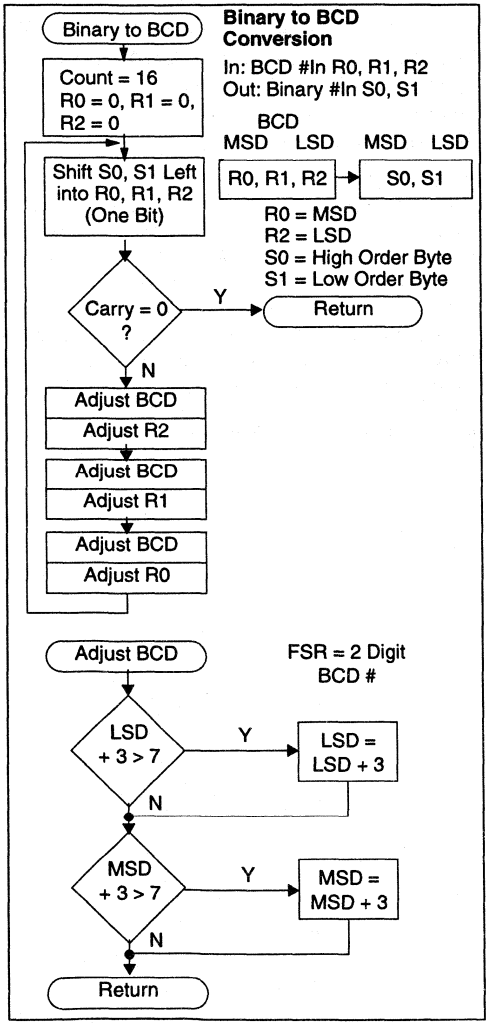
BINARY TO BCD CONVERSION

Two routines are provided: one routine converts a 16-bit binary number to a five-digit BCD number and the other routine converts an 8-bit binary number to a two-digit BCD number. The listing of these routines are given in Appendices G and H. The performance specs for the routines are shown below:

TABLE 7: PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
Binary (8-Bit) to BCD	10	81 (Worst Case)
Binary (16-Bit) to BCD	30	719

FIGURE 2: Flowchart for Binary to BCD Conversion



BCD ADDITION & SUBTRACTION

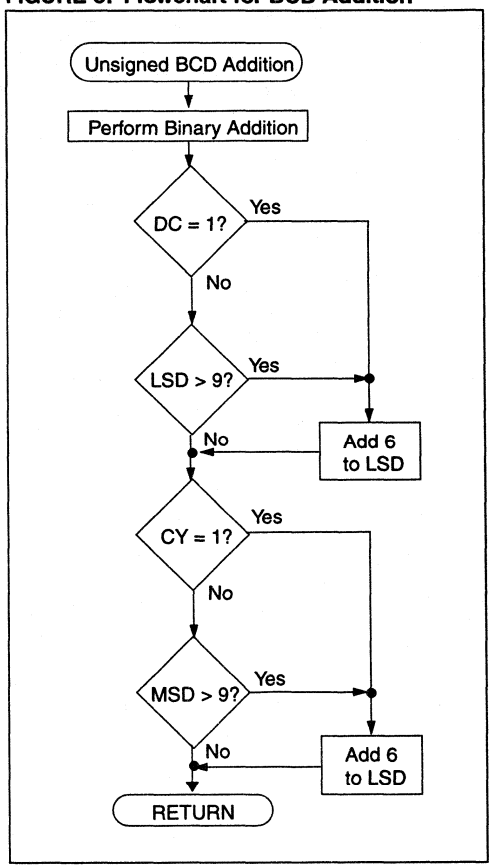
These two routines perform a two-digit unsigned BCD addition and subtraction. The results are the sum (or difference) in one file register and with a overflow carry-bit in another file register. The performance specs for the routines are shown below:

TABLE 8: PERFORMANCE SPECS

Spec	Program Memory	Instruction Cycles
BCD Addition	29	23 (Worst Case)
BCD Subtraction	31	21 (Worst Case)

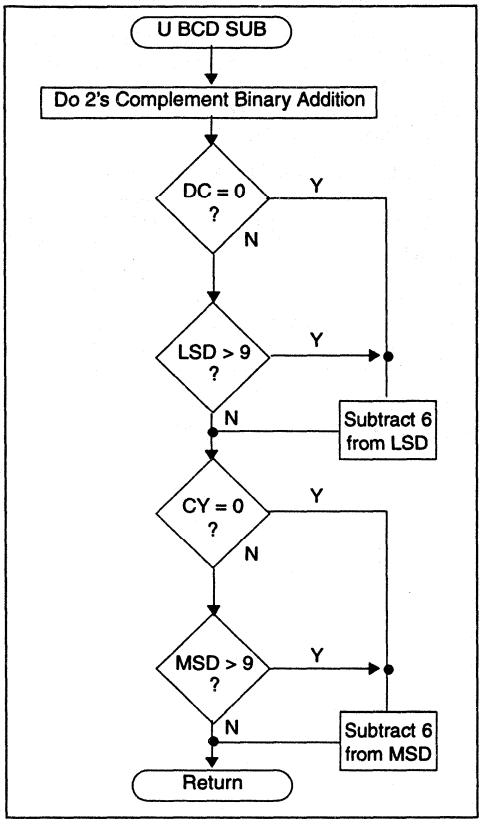
The listing files for the above two routines are given in Appendices J and K. The flow charts for BCD addition and BCD subtraction are given in Figure 3 and Figure 4, respectively.

FIGURE 3: Flowchart for BCD Addition



5

FIGURE 4: Flowchart for BCD Subtraction



SQUARE ROOT

Often in many applications one needs to find the square root of a number. Of many numerical methods to find the square root of a number, the Newton-Raphson method is very attractive because of its fast convergence rate. In this method the square root of a number, "N", is obtained from the approximate solution of:

$$f(Y) = Y^2 - N = 0$$

The function "f(Y)" can be expanded about Y_0 using first order Taylor polynomial expansion as:

Equation 1: $f(Y)=f(Y0)+(Y-Y0)f'(Y0)+(Y-Y0)2f''(Y0)/2!+....$

If X is a root of f(Y), then f(X) = 0:

$f(X) = f(Y0) + (x - Y0)\ f'(Y0) + (X - Y0)\ 2f''(Y0) / 2! + ... = 0$

If Y0 is an approximate root of f(Y), then higher order terms are negligible. Therefore:

Equation 2: $f(Y_0) + (X - Y_0)\ f'(Y_0)$ [i.e., $X = Y0 - f(Y0) / f'(Y0)$]

Thus, X is a better approximation for Y0. From this, the sequence {Xn} can be generated:

Equation 3: $Xn = Xn - 1 - f(Xn - 1) / f'(Xn - 1),\ n \geq 1$

From equation 1 and equation 3 we get,

Equation 4: $Xn = 0.5^* \{Xn - 1 + N/Xn - 1\}$

The initial approximate root of N is taken to be N/2. If the approximate range of N is known a priori, then the total number of iterations may be cut down by starting with a better approximate root than N/2.

This program, as listed in Appendix K, computes the square root of a 16-bit number. This routine uses double precision math routines (division and addition) as described in the previous pages of this application note. The divide routines are integrated into the source listing. For fixed point divide routines, see Appendices L - O.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40 Released         MULT8X8S.ASM   1-16-1997  12:54:42         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;                    8x8 Software Multiplier
                     00005 ;                ( Code Efficient : Looped Code )
                     00006 ;*****************************************************************
                     00007 ;
                     00008 ;   The 16 bit result is stored in 2 bytes
                     00009 ;
                     00010 ; Before calling the subroutine " mpy ", the multiplier should
                     00011 ; be loaded in location " mulplr ", and the multiplicand in
                     00012 ; " mulcnd " . The 16 bit result is stored in locations
                     00013 ; H_byte & L_byte.
                     00014 ;
                     00015 ;       Performance :
                     00016 ;                          Program Memory  :  15 locations
                     00017 ;                          # of cycles     :  71
                     00018 ;                          Scratch RAM     :   0 locations
                     00019 ;
                     00020 ;
                     00021 ;       Program:           MULT8x8S.ASM
                     00022 ;       Revision Date:
                     00023 ;                          1-13-97    Compatibility with MPASMWIN 1.40
                     00024 ;
                     00025 ; This routine is optimized for code efficiency (looped code)
                     00026 ; For time efficiency code refer to "mult8x8F.asm"(straight line code)
                     00027 ;*****************************************************************
                     00028 ;
  00000009           00029 mulcnd  equ     09      ; 8 bit multiplicand
  00000010           00030 mulplr  equ     10      ; 8 bit multiplier
  00000012           00031 H_byte  equ     12      ; High byte of the 16 bit result
  00000013           00032 L_byte  equ     13      ; Low byte of the 16 bit result
  00000014           00033 count   equ     14      ; loop counter
                     00034 ;
                     00035 ;
                     00036         include         "p16c5x.inc"
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology, Inc.
                     00224         LIST
                     00037
  00000001           00038 Same    equ     1
                     00039
                     00040 ;
                     00041 ; ****************************          Begin Multiplier Routine
0000 0072            00042 mpy_S   clrf    H_byte
0001 0073            00043         clrf    L_byte
0002 0C08            00044         movlw   8
0003 0034            00045         movwf   count
0004 0209            00046         movf    mulcnd,W
0005 0403            00047         bcf     STATUS,C    ; Clear the carry bit in the status Reg.
0006 0330            00048 loop    rrf     mulplr, F
0007 0603            00049         btfsc   STATUS,C
0008 01F2            00050         addwf   H_byte,Same
0009 0332            00051         rrf     H_byte,Same
```


© 1997 Microchip Technology Inc.

```
000A 0333           00052         rrf     L_byte,Same
000B 02F4           00053         decfsz  count, F
000C 0A06           00054         goto    loop
                    00055 ;
000D 0800           00056         retlw   0
                    00057 ;
                    00058 ;*****************************************************************
                    00059 ;               Test Program
                    00060 ;*****************************************************************
000E 0CFF           00061 main    movlw   0FF
000F 0030           00062         movwf   mulplr        ; multiplier (in mulplr) = 0FF
0010 0CFF           00063         movlw   0FF           ; multiplicand(W Reg )   = 0FF
0011 0029           00064         movwf   mulcnd
                    00065 ;
0012 0900           00066         call    mpy_S         ; The result 0FF*0FF = FE01 is in locations
                    00067 ;                             ; H_byte & L_byte
                    00068 ;
0013 0A13           00069 self    goto    self
                    00070 ;
01FF                00071         org     01FF
01FF 0A0E           00072         goto    main
                    00073 ;
                    00074         END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXX------------ ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:    21
Program Memory Words Free:   491

Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B:

```
MPASM 01.40 Released         MULT8X8F.ASM   1-16-1997  12:54:10         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;                    8x8 Software Multiplier
                     00005 ;                ( Fast Version : Straight Line Code )
                     00006 ;*****************************************************************
                     00007 ;
                     00008 ;   The 16 bit result is stored in 2 bytes
                     00009 ;
                     00010 ; Before calling the subroutine " mpy ", the multiplier should
                     00011 ; be loaded in location " mulplr ", and the multiplicand in
                     00012 ; " mulcnd " . The 16 bit result is stored in locations
                     00013 ; H_byte & L_byte.
                     00014 ;
                     00015 ;       Performance :
                     00016 ;                          Program Memory  :  35 locations
                     00017 ;                          # of cycles     :  37
                     00018 ;                          Scratch RAM     :   0 locations
                     00019 ;
                     00020 ;
                     00021 ;       Program:           MULT8x8F.ASM
                     00022 ;       Revision Date:
                     00023 ;                          1-13-97      Compatibility with MPASMWIN 1.40
                     00024 ;
                     00025 ; This routine is optimized for speed efficiency (straight line code)
                     00026 ; For code efficiency, refer to "mult8x8S.asm" (looped code)
                     00027 ;*****************************************************************
                     00028 ;
  00000009           00029 mulcnd  equ     09      ; 8 bit multiplicand
  00000010           00030 mulplr  equ     10      ; 8 bit multiplier
  00000012           00031 H_byte  equ     12      ; High byte of the 16 bit result
  00000013           00032 L_byte  equ     13      ; Low byte of the 16 bit result
                     00033 ;
                     00034 ;
                     00035         include         "p16c5x.inc"
                     00001         LIST
                     00002 ; P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology, Inc.
                     00224         LIST
                     00036
  00000001           00037 Same    equ     1
                     00038
                     00039 ;
                     00040 ;****   Define a macro for adding & right shifting  **
                     00041 ;
                     00042 mult    MACRO   bit          ; Begin macro
                     00043         btfsc   mulplr,bit
                     00044         addwf   H_byte,Same
                     00045         rrf     H_byte,Same
                     00046         rrf     L_byte,Same
                     00047         ENDM                 ; End of macro
                     00048 ;
                     00049 ; ****************************  Begin Multiplier Routine
0000 0072            00050 mpy_F clrf    H_byte
0001 0073            00051       clrf    L_byte
```

```
0002 0209           00052       movf    mulcnd,W   ; move the multiplicand to W reg.
0003 0403           00053       bcf     STATUS,C   ; Clear the carry bit in the status Reg.
                    00054       mult    0
0004 0610               M       btfsc   mulplr,0
0005 01F2               M       addwf   H_byte,Same
0006 0332               M       rrf     H_byte,Same
0007 0333               M       rrf     L_byte,Same
                    00055       mult    1
0008 0630               M       btfsc   mulplr,1
0009 01F2               M       addwf   H_byte,Same
000A 0332               M       rrf     H_byte,Same
000B 0333               M       rrf     L_byte,Same
                    00056       mult    2
000C 0650               M       btfsc   mulplr,2
000D 01F2               M       addwf   H_byte,Same
000E 0332               M       rrf     H_byte,Same
000F 0333               M       rrf     L_byte,Same
                    00057       mult    3
0010 0670               M       btfsc   mulplr,3
0011 01F2               M       addwf   H_byte,Same
0012 0332               M       rrf     H_byte,Same
0013 0333               M       rrf     L_byte,Same
                    00058       mult    4
0014 0690               M       btfsc   mulplr,4
0015 01F2               M       addwf   H_byte,Same
0016 0332               M       rrf     H_byte,Same
0017 0333               M       rrf     L_byte,Same
                    00059       mult    5
0018 06B0               M       btfsc   mulplr,5
0019 01F2               M       addwf   H_byte,Same
001A 0332               M       rrf     H_byte,Same
001B 0333               M       rrf     L_byte,Same
                    00060       mult    6
001C 06D0               M       btfsc   mulplr,6
001D 01F2               M       addwf   H_byte,Same
001E 0332               M       rrf     H_byte,Same
001F 0333               M       rrf     L_byte,Same
                    00061       mult    7
0020 06F0               M       btfsc   mulplr,7
0021 01F2               M       addwf   H_byte,Same
0022 0332               M       rrf     H_byte,Same
0023 0333               M       rrf     L_byte,Same
                    00062 ;
0024 0800           00063       retlw   0
                    00064 ;
                    00065 ;*****************************************************************
                    00066 ;               Test Program
                    00067 ;*****************************************************************
0025 0CFF           00068 main  movlw   0FF
0026 0030           00069       movwf   mulplr    ; multiplier (in mulplr)  = 0FF
0027 0CFF           00070       movlw   0FF
0028 0029           00071       movwf   mulcnd    ; multiplicand(in mulcnd) = 0FF
                    00072 ;
0029 0900           00073       call    mpy_F     ; The result 0FF*0FF = FE01 is in locations
                    00074 ;                       ; H_byte & L_byte
                    00075 ;
002A 0A2A           00076 self goto     self
                    00077 ;
01FF                00078       org     01FF
01FF 0A25           00079       goto    main
                    00080 ;
                    00081         END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXX----- ----------------
```


© 1997 Microchip Technology Inc.

```
01C0 : ---------------- ---------------- ---------------- ----------------X

All other memory blocks unused.

Program Memory Words Used:     44
Program Memory Words Free:    468


Errors    :      0
Warnings  :      0 reported,      0 suppressed
Messages  :      0 reported,      0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C:DOUBLE PRECISION MULTIPLICATION LISTING (LOOPED)

```
MPASM 01.40 Released          DBL_MPYS.ASM   1-16-1997  12:53:00          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54,  n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;                     Double Precision Multiplication
                     00005 ;
                     00006 ;               ( Optimized for Code Size : Looped Code )
                     00007 ;
                     00008 ;*****************************************************************;
                     00009 ; Multiplication: ACCb(16 bits)*ACCa(16 bits) -> ACCb,ACCc (32 bits)
                     00010 ;   (a) Load the 1st operand in location ACCaLO & ACCaHI (16 bits)
                     00011 ;   (b) Load the 2nd operand in location ACCbLO & ACCbHI (16 bits)
                     00012 ;   (c) CALL D_mpy
                     00013 ;   (d) The 32 bit result is in location (ACCbHI,ACCbLO,ACCcHI,ACCcLO)
                     00014 ;
                     00015 ;    Performance :
                     00016 ;                 Program Memory  :   033
                     00017 ;                 Clock Cycles    :   333
                     00018 ;
                     00019 ;     Note : The above timing is the worst case timing, when the
                     00020 ;            register ACCb = FFFF. The speed may be improved if
                     00021 ;            the register ACCb contains a number ( out of the two
                     00022 ;            numbers ) with less number of 1s.
                     00023 ;            The performance specs are for Unsigned arithmetic (i.e,
                     00024 ;            with "SIGNED equ  FALSE").
                     00025 ;
                     00026 ;            The performance specs are for Unsigned arithmetic (i.e,
                     00027 ;            with "SIGNED equ  FALSE").
                     00028 ;
                     00029 ;
                     00030 ;         Program:        DBL_MPYS.ASM
                     00031 ;         Revision Date:
                     00032 ;                         1-13-97   Compatibility with MPASMWIN 1.40
                     00033 ;
                     00034 ;*****************************************************************;
                     00035 ;
00000010             00036 ACCaLO  equ     0x10
00000011             00037 ACCaHI  equ     0x11
00000012             00038 ACCbLO  equ     0x12
00000013             00039 ACCbHI  equ     0x13
00000014             00040 ACCcLO  equ     0x14
00000015             00041 ACCcHI  equ     0x15
00000016             00042 ACCdLO  equ     0x16
00000017             00043 ACCdHI  equ     0x17
00000018             00044 temp    equ     0x18
00000019             00045 sign    equ     0x19
0000001F             00046 Flags   equ     0x1F
                     00047 ;
                     00048         include "p16c5x.inc"
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology, Inc.
                     00224         LIST
                     00049
000001FF             00050 PIC54   equ     1FFH    ; Define Reset Vector
00000001             00051 TRUE    equ     1
```


© 1997 Microchip Technology Inc.

```
  00000000          00052 FALSE   equ     0
  00000007          00053 MSB     equ     7
                    00054
0000                00055         org     0
                    00056 ;*****************************************************************
  00000001          00057 SIGNED  equ     TRUE            ; Set This To 'TRUE' if the routines
                    00058 ;                               ; for Multiplication & Division needs
                    00059 ;                               ; to be assembled as Signed Integer
                    00060 ;                               ; Routines. If 'FALSE' the above two
                    00061 ;                               ; routines ( D_mpy & D_div ) use
                    00062 ;                               ; unsigned arithmetic.
                    00063 ;*****************************************************************
                    00064 ;         Double Precision Addition ( ACCb + ACCa -> ACCb )
                    00065 ;
0000 041F           00066 D_add   bcf     Flags,C ;Clear temp Carry bit
0001 0210           00067         movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
0002 01F2           00068         addwf   ACCbLO, F       ;add lsb
0003 0603           00069         btfsc   STATUS,C        ;add in carry
0004 02B3           00070         incf    ACCbHI, F
0005 0603           00071         btfsc   STATUS,C
0006 051F           00072         bsf     Flags,C
0007 0211           00073         movf    ACCaHI,W
0008 01F3           00074         addwf   ACCbHI, F         ;add msb
0009 061F           00075         btfsc   Flags,C
000A 0503           00076         bsf     STATUS,C
000B 0800           00077         retlw   0
                    00078 ;*****************************************************************
                    00079 ;               Double Precision Multiply ( 16x16 -> 32 )
                    00080 ;         ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
                    00081 ;  in ACCb ( ACCbHI,ACCbLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
                    00082 ;
000C                00083 D_mpyS                          ;results in ACCb(16 msb's) and ACCc(16
lsb's)
                    00084 ;
                    00085       IF   SIGNED
000C 0935           00086       CALL    S_SIGN
                    00087       ENDIF
                    00088 ;
000D 0926           00089         call    setup
000E 0337           00090 mloop   rrf     ACCdHI, F       ;rotate d right
000F 0336           00091         rrf     ACCdLO, F
0010 0603           00092         btfsc   STATUS,C        ;need to add?
0011 0900           00093         call    D_add
0012 0333           00094         rrf     ACCbHI, F
0013 0332           00095         rrf     ACCbLO, F
0014 0335           00096         rrf     ACCcHI, F
0015 0334           00097         rrf     ACCcLO, F
0016 02F8           00098         decfsz  temp, F         ;loop until all bits checked
0017 0A0E           00099         goto    mloop
                    00100 ;
                    00101     IF    SIGNED
0018 07F9           00102         btfss   sign,MSB
0019 0800           00103         retlw   0
001A 0274           00104         comf    ACCcLO, F       ; negate ACCa ( -ACCa -> ACCa )
001B 02B4           00105         incf    ACCcLO, F
001C 0643           00106         btfsc   STATUS,Z
001D 00F5           00107         decf    ACCcHI, F
001E 0275           00108         comf    ACCcHI, F
001F 0643           00109         btfsc   STATUS,Z
0020 0272           00110 neg_B   comf    ACCbLO, F       ; negate ACCb
0021 02B2           00111         incf    ACCbLO, F
0022 0643           00112         btfsc   STATUS,Z
0023 00F3           00113         decf    ACCbHI, F
0024 0273           00114         comf    ACCbHI, F
0025 0800           00115         retlw   0
                    00116     ELSE
```


© 1997 Microchip Technology Inc.

```
                00117         retlw   0
                00118     ENDIF
                00119 ;
                00120 ;*****************************************************************
                00121 ;
0026 0C10       00122 setup   movlw   .16             ; for 16 shifts
0027 0038       00123         movwf   temp
0028 0213       00124         movf    ACCbHI,W        ; move ACCb to ACCd
0029 0037       00125         movwf   ACCdHI
002A 0212       00126         movf    ACCbLO,W
002B 0036       00127         movwf   ACCdLO
002C 0073       00128         clrf    ACCbHI
002D 0072       00129         clrf    ACCbLO
002E 0800       00130         retlw   0
                00131 ;
                00132 ;*****************************************************************
                00133 ;
002F 0270       00134 neg_A   comf    ACCaLO, F       ; negate ACCa ( -ACCa -> ACCa )
0030 02B0       00135         incf    ACCaLO, F
0031 0643       00136         btfsc   STATUS,Z
0032 00F1       00137         decf    ACCaHI, F
0033 0271       00138         comf    ACCaHI, F
0034 0800       00139         retlw   0
                00140 ;
                00141 ;*****************************************************************
                00142 ;  Assemble this section only if Signed Arithmetic Needed
                00143 ;
                00144     IF    SIGNED
                00145 ;
0035 0211       00146 S_SIGN  movf    ACCaHI,W
0036 0193       00147         xorwf   ACCbHI,W
0037 0039       00148         movwf   sign
0038 07F3       00149         btfss   ACCbHI,MSB        ; if MSB set go & negate ACCb
0039 0A3F       00150         goto    chek_A
                00151 ;
003A 0272       00152         comf    ACCbLO, F         ; negate ACCb
003B 02B2       00153         incf    ACCbLO, F
003C 0643       00154         btfsc   STATUS,Z
003D 00F3       00155         decf    ACCbHI, F
003E 0273       00156         comf    ACCbHI, F
                00157 ;
003F 07F1       00158 chek_A  btfss   ACCaHI,MSB        ; if MSB set go & negate ACCa
0040 0800       00159         retlw   0
0041 0A2F       00160         goto    neg_A
                00161 ;
                00162     ENDIF
                00163 ;
                00164 ;*****************************************************************
                00165 ;                         Test Program
                00166 ;*****************************************************************
                00167 ;    Load constant values to ACCa & ACCb for testing
                00168 ;
0042 0C01       00169 main    movlw   1
0043 0031       00170         movwf   ACCaHI
0044 0CFF       00171         movlw   0FF             ; loads ACCa = 01FF
0045 0030       00172         movwf   ACCaLO
                00173 ;
0046 0C7F       00174         movlw   0x7F
0047 0033       00175         movwf   ACCbHI
0048 0CFF       00176         movlw   0xFF            ; loads ACCb = 7FFF
0049 0032       00177         movwf   ACCbLO
                00178 ;
004A 090C       00179         call    D_mpyS          ; Here (ACCb,ACCc) = 00FF 7E01
                00180 ;
004B 0A4B       00181 self    goto    self
                00182 ;
```


© 1997 Microchip Technology Inc.

```
01FF                00183         org     PIC54
01FF 0A42           00184         goto    main
                    00185         END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)


0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXX---- ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:    77
Program Memory Words Free:   435


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

5

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX D:DOUBLE PRECISION MULTIPLICATION LISTINGS (FAST)

```
MPASM 01.40 Released         DBL_MPYF.ASM   1-16-1997  12:52:26         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;                       Double Precision Multiplication
                     00005 ;
                     00006 ;               ( Optimized for Speed : straight Line Code )
                     00007 ;
                     00008 ;*****************************************************************;
                     00009 ;Multiplication : ACCb(16 bits) * ACCa(16 bits) -> ACCb,ACCc (32 bits)
                     00010 ;   (a) Load the 1st operand in location ACCaLO & ACCaHI (16 bits)
                     00011 ;   (b) Load the 2nd operand in location ACCbLO & ACCbHI (16 bits)
                     00012 ;   (c) CALL D_mpy
                     00013 ;   (d) The 32 bit result is in location (ACCbHI,ACCbLO,ACCcHI,ACCcLO)
                     00014 ;
                     00015 ;    Performance :
                     00016 ;                 Program Memory  :       240
                     00017 ;                 Clock Cycles    :       233
                     00018 ;
                     00019 ;    Note : The above timing is the worst case timing, when the
                     00020 ;           register ACCb = FFFF. The speed may be improved if
                     00021 ;           the register ACCb contains a number (out of the two
                     00022 ;           numbers) with less number of 1s.
                     00023 ;
                     00024 ;           The performance specs are for Unsigned arithmetic (i.e,
                     00025 ;           with "SIGNED equ  FALSE").
                     00026 ;
                     00027 ;        Program:        DBL_MPYF.ASM
                     00028 ;        Revision Date:
                     00029 ;                        1-13-97  Compatibility with MPASMWIN 1.40
                     00030 ;
                     00031 ;*****************************************************************;
                     00032 ;
  00000010           00033 ACCaLO  equ     10
  00000011           00034 ACCaHI  equ     11
  00000012           00035 ACCbLO  equ     12
  00000013           00036 ACCbHI  equ     13
  00000014           00037 ACCcLO  equ     14
  00000015           00038 ACCcHI  equ     15
  00000016           00039 ACCdLO  equ     16
  00000017           00040 ACCdHI  equ     17
  00000018           00041 temp    equ     18
  00000019           00042 sign    equ     19
                     00043 ;
                     00044         include "p16c5x.inc"
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology, Inc.
                     00224         LIST
                     00045
  000001FF           00046 PIC54   equ     1FFH    ; Define Reset Vector
  00000001           00047 TRUE    equ     1
  00000000           00048 FALSE   equ     0
                     00049
0000                 00050         org     0
                     00051 ;*****************************************************************
```


© 1997 Microchip Technology Inc.

```
 00000000      00052 SIGNED  equ     FALSE           ; Set This To 'TRUE' if the routines
               00053 ;                               ; for Multiplication & Division needs
               00054 ;                               ; to be assembled as Signed Integer
               00055 ;                               ; Routines. If 'FALSE' the above two
               00056 ;                               ; routines ( D_mpy & D_div ) use
               00057 ;                               ; unsigned arithmetic.
               00058 ;*****************************************************************
               00059 ;       multiplication macro
               00060 ;
               00061 mulMac  MACRO
               00062         LOCAL   NO_ADD
               00063 ;
               00064         rrf     ACCdHI, F       ;rotate d right
               00065         rrf     ACCdLO, F
               00066         btfss   STATUS,C        ; need to add?
               00067         goto    NO_ADD          ; no addition necessary
               00068         movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
               00069         addwf   ACCbLO, F       ;add lsb
               00070         btfsc   STATUS,C        ; add in carry
               00071         incf    ACCbHI, F
               00072         movf    ACCaHI,W
               00073         addwf   ACCbHI, F       ;add msb
               00074 NO_ADD  rrf     ACCbHI, F
               00075         rrf     ACCbLO, F
               00076         rrf     ACCcHI, F
               00077         rrf     ACCcLO, F
               00078 ;
               00079         ENDM
               00080 ;
               00081 ;*****************************************************************;
               00082 ;               Double Precision Multiply ( 16x16 -> 32 )
               00083 ;         ( ACCb*ACCa -> ACCb,ACCc ) : 32 bit output with high word
               00084 ;  in ACCb ( ACCbHI,ACCbLO ) and low word in ACCc ( ACCcHI,ACCcLO ).
               00085 ;
 0000          00086 D_mpyF              ;results in ACCb(16 msb's) and ACCc(16 lsb's)
               00087 ;
               00088       IF   SIGNED
               00089       CALL    S_SIGN
               00090       ENDIF
               00091 ;
 0000 09E2     00092         call    setup
               00093 ;
               00094 ; use the mulMac macro 16 times
               00095 ;
               00096         mulMac
   0000           M         LOCAL   NO_ADD
                  M ;
 0001 0337        M         rrf     ACCdHI, F       ;rotate d right
 0002 0336        M         rrf     ACCdLO, F
 0003 0703        M         btfss   STATUS,C        ; need to add?
 0004 0A0B        M         goto    NO_ADD          ; no addition necessary
 0005 0210        M         movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
 0006 01F2        M         addwf   ACCbLO, F       ; add lsb
 0007 0603        M         btfsc   STATUS,C        ; add in carry
 0008 02B3        M         incf    ACCbHI, F
 0009 0211        M         movf    ACCaHI,W
 000A 01F3        M         addwf   ACCbHI, F       ; add msb
 000B 0333        M NO_ADD  rrf     ACCbHI, F
 000C 0332        M         rrf     ACCbLO, F
 000D 0335        M         rrf     ACCcHI, F
 000E 0334        M         rrf     ACCcLO, F
                  M ;
                00097         mulMac
   0000           M         LOCAL   NO_ADD
                  M ;
 000F 0337        M         rrf     ACCdHI, F       ; rotate d right
```


© 1997 Microchip Technology Inc.

```
0010 0336           M          rrf     ACCdLO, F
0011 0703           M          btfss   STATUS,C        ; need to add?
0012 0A19           M          goto    NO_ADD          ; no addition necessary
0013 0210           M          movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
0014 01F2           M          addwf   ACCbLO, F       ;add lsb
0015 0603           M          btfsc   STATUS,C        ; add in carry
0016 02B3           M          incf    ACCbHI, F
0017 0211           M          movf    ACCaHI,W
0018 01F3           M          addwf   ACCbHI, F       ; add msb
0019 0333           M NO_ADD   rrf     ACCbHI, F
001A 0332           M          rrf     ACCbLO, F
001B 0335           M          rrf     ACCcHI, F
001C 0334           M          rrf     ACCcLO, F
                    M ;
                  00098        mulMac
  0000              M          LOCAL   NO_ADD
                    M ;
001D 0337           M          rrf     ACCdHI, F       ; rotate d right
001E 0336           M          rrf     ACCdLO, F
001F 0703           M          btfss   STATUS,C        ; need to add?
0020 0A27           M          goto    NO_ADD          ; no addition necessary
0021 0210           M          movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
0022 01F2           M          addwf   ACCbLO, F       ; add lsb
0023 0603           M          btfsc   STATUS,C        ; add in carry
0024 02B3           M          incf    ACCbHI, F
0025 0211           M          movf    ACCaHI,W
0026 01F3           M          addwf   ACCbHI, F       ; add msb
0027 0333           M NO_ADD   rrf     ACCbHI, F
0028 0332           M          rrf     ACCbLO, F
0029 0335           M          rrf     ACCcHI, F
002A 0334           M          rrf     ACCcLO, F
                    M ;
                  00099        mulMac
  0000              M          LOCAL   NO_ADD
                    M ;
002B 0337           M          rrf     ACCdHI, F       ; rotate d right
002C 0336           M          rrf     ACCdLO, F
002D 0703           M          btfss   STATUS,C        ; need to add?
002E 0A35           M          goto    NO_ADD          ; no addition necessary
002F 0210           M          movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
0030 01F2           M          addwf   ACCbLO, F       ; add lsb
0031 0603           M          btfsc   STATUS,C        ; add in carry
0032 02B3           M          incf    ACCbHI, F
0033 0211           M          movf    ACCaHI,W
0034 01F3           M          addwf   ACCbHI, F       ; add msb
0035 0333           M NO_ADD   rrf     ACCbHI, F
0036 0332           M          rrf     ACCbLO, F
0037 0335           M          rrf     ACCcHI, F
0038 0334           M          rrf     ACCcLO, F
                    M ;
                  00100        mulMac
  0000              M          LOCAL   NO_ADD
                    M ;
0039 0337           M          rrf     ACCdHI, F       ; rotate d right
003A 0336           M          rrf     ACCdLO, F
003B 0703           M          btfss   STATUS,C        ; need to add?
003C 0A43           M          goto    NO_ADD          ; no addition necessary
003D 0210           M          movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
003E 01F2           M          addwf   ACCbLO, F       ; add lsb
003F 0603           M          btfsc   STATUS,C        ; add in carry
0040 02B3           M          incf    ACCbHI, F
0041 0211           M          movf    ACCaHI,W
0042 01F3           M          addwf   ACCbHI, F       ; add msb
0043 0333           M NO_ADD   rrf     ACCbHI, F
0044 0332           M          rrf     ACCbLO, F
0045 0335           M          rrf     ACCcHI, F
```


© 1997 Microchip Technology Inc.

```
0046 0334           M         rrf     ACCcLO, F
                    M ;
                  00101       mulMac
  0000              M         LOCAL   NO_ADD
                    M ;
0047 0337           M         rrf     ACCdHI, F       ;rotate d right
0048 0336           M         rrf     ACCdLO, F
0049 0703           M         btfss   STATUS,C        ; need to add?
004A 0A51           M         goto    NO_ADD          ; no addition necessary
004B 0210           M         movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
004C 01F2           M         addwf   ACCbLO, F       ; add lsb
004D 0603           M         btfsc   STATUS,C        ; add in carry
004E 02B3           M         incf    ACCbHI, F
004F 0211           M         movf    ACCaHI,W
0050 01F3           M         addwf   ACCbHI, F       ; add msb
0051 0333           M NO_ADD  rrf     ACCbHI, F
0052 0332           M         rrf     ACCbLO, F
0053 0335           M         rrf     ACCcHI, F
0054 0334           M         rrf     ACCcLO, F
                    M ;
                  00102       mulMac
  0000              M         LOCAL   NO_ADD
                    M ;
0055 0337           M         rrf     ACCdHI, F       ; rotate d right
0056 0336           M         rrf     ACCdLO, F
0057 0703           M         btfss   STATUS,C        ; need to add?
0058 0A5F           M         goto    NO_ADD          ; no addition necessary
0059 0210           M         movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
005A 01F2           M         addwf   ACCbLO, F       ; add lsb
005B 0603           M         btfsc   STATUS,C        ; add in carry
005C 02B3           M         incf    ACCbHI, F
005D 0211           M         movf    ACCaHI,W
005E 01F3           M         addwf   ACCbHI, F       ; add msb
005F 0333           M NO_ADD  rrf     ACCbHI, F
0060 0332           M         rrf     ACCbLO, F
0061 0335           M         rrf     ACCcHI, F
0062 0334           M         rrf     ACCcLO, F
                    M ;
                  00103       mulMac
  0000              M         LOCAL   NO_ADD
                    M ;
0063 0337           M         rrf     ACCdHI, F       ; rotate d right
0064 0336           M         rrf     ACCdLO, F
0065 0703           M         btfss   STATUS,C        ; need to add?
0066 0A6D           M         goto    NO_ADD          ; no addition necessary
0067 0210           M         movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
0068 01F2           M         addwf   ACCbLO, F       ; add lsb
0069 0603           M         btfsc   STATUS,C        ; add in carry
006A 02B3           M         incf    ACCbHI, F
006B 0211           M         movf    ACCaHI,W
006C 01F3           M         addwf   ACCbHI, F       ; add msb
006D 0333           M NO_ADD  rrf     ACCbHI, F
006E 0332           M         rrf     ACCbLO, F
006F 0335           M         rrf     ACCcHI, F
0070 0334           M         rrf     ACCcLO, F
                    M ;
                  00104       mulMac
  0000              M         LOCAL   NO_ADD
                    M ;
0071 0337           M         rrf     ACCdHI, F       ; rotate d right
0072 0336           M         rrf     ACCdLO, F
0073 0703           M         btfss   STATUS,C        ; need to add?
0074 0A7B           M         goto    NO_ADD          ; no addition necessary
0075 0210           M         movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
0076 01F2           M         addwf   ACCbLO, F       ; add lsb
0077 0603           M         btfsc   STATUS,C        ; add in carry
```


© 1997 Microchip Technology Inc.

```
0078 02B3          M          incf    ACCbHI, F
0079 0211          M          movf    ACCaHI,W
007A 01F3          M          addwf   ACCbHI, F       ; add msb
007B 0333          M NO_ADD   rrf     ACCbHI, F
007C 0332          M          rrf     ACCbLO, F
007D 0335          M          rrf     ACCcHI, F
007E 0334          M          rrf     ACCcLO, F
                   M ;
                 00105        mulMac
  0000             M          LOCAL   NO_ADD
                   M ;
007F 0337          M          rrf     ACCdHI, F       ; rotate d right
0080 0336          M          rrf     ACCdLO, F
0081 0703          M          btfss   STATUS,C        ; need to add?
0082 0A89          M          goto    NO_ADD          ; no addition necessary
0083 0210          M          movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
0084 01F2          M          addwf   ACCbLO, F       ; add lsb
0085 0603          M          btfsc   STATUS,C        ;  add in carry
0086 02B3          M          incf    ACCbHI, F
0087 0211          M          movf    ACCaHI,W
0088 01F3          M          addwf   ACCbHI, F       ; add msb
0089 0333          M NO_ADD   rrf     ACCbHI, F
008A 0332          M          rrf     ACCbLO, F
008B 0335          M          rrf     ACCcHI, F
008C 0334          M          rrf     ACCcLO, F
                   M ;
                 00106        mulMac
  0000             M          LOCAL   NO_ADD
                   M ;
008D 0337          M          rrf     ACCdHI, F       ; rotate d right
008E 0336          M          rrf     ACCdLO, F
008F 0703          M          btfss   STATUS,C        ; need to add?
0090 0A97          M          goto    NO_ADD          ; no addition necessary
0091 0210          M          movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
0092 01F2          M          addwf   ACCbLO, F       ; add lsb
0093 0603          M          btfsc   STATUS,C        ; add in carry
0094 02B3          M          incf    ACCbHI, F
0095 0211          M          movf    ACCaHI,W
0096 01F3          M          addwf   ACCbHI, F       ; add msb
0097 0333          M NO_ADD   rrf     ACCbHI, F
0098 0332          M          rrf     ACCbLO, F
0099 0335          M          rrf     ACCcHI, F
009A 0334          M          rrf     ACCcLO, F
                   M ;
                 00107        mulMac
  0000             M          LOCAL   NO_ADD
                   M ;
009B 0337          M          rrf     ACCdHI, F       ; rotate d right
009C 0336          M          rrf     ACCdLO, F
009D 0703          M          btfss   STATUS,C        ; need to add?
009E 0AA5          M          goto    NO_ADD          ; no addition necessary
009F 0210          M          movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
00A0 01F2          M          addwf   ACCbLO, F       ; add lsb
00A1 0603          M          btfsc   STATUS,C        ; add in carry
00A2 02B3          M          incf    ACCbHI, F
00A3 0211          M          movf    ACCaHI,W
00A4 01F3          M          addwf   ACCbHI, F       ; add msb
00A5 0333          M NO_ADD   rrf     ACCbHI, F
00A6 0332          M          rrf     ACCbLO, F
00A7 0335          M          rrf     ACCcHI, F
00A8 0334          M          rrf     ACCcLO, F
                   M ;
                 00108        mulMac
  0000             M          LOCAL   NO_ADD
                   M ;
00A9 0337          M          rrf     ACCdHI, F       ; rotate d right
```


© 1997 Microchip Technology Inc.

```
00AA 0336           M          rrf     ACCdLO, F
00AB 0703           M          btfss   STATUS,C        ; need to add?
00AC 0AB3           M          goto    NO_ADD          ; no addition necessary
00AD 0210           M          movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
00AE 01F2           M          addwf   ACCbLO, F       ; add lsb
00AF 0603           M          btfsc   STATUS,C        ; add in carry
00B0 02B3           M          incf    ACCbHI, F
00B1 0211           M          movf    ACCaHI,W
00B2 01F3           M          addwf   ACCbHI, F       ; add msb
00B3 0333           M NO_ADD   rrf     ACCbHI, F
00B4 0332           M          rrf     ACCbLO, F
00B5 0335           M          rrf     ACCcHI, F
00B6 0334           M          rrf     ACCcLO, F
                    M ;
                  00109        mulMac
  0000              M          LOCAL   NO_ADD
                    M ;
00B7 0337           M          rrf     ACCdHI, F       ; rotate d right
00B8 0336           M          rrf     ACCdLO, F
00B9 0703           M          btfss   STATUS,C        ; need to add?
00BA 0AC1           M          goto    NO_ADD          ; no addition necessary
00BB 0210           M          movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
00BC 01F2           M          addwf   ACCbLO, F       ; add lsb
00BD 0603           M          btfsc   STATUS,C        ; add in carry
00BE 02B3           M          incf    ACCbHI, F
00BF 0211           M          movf    ACCaHI,W
00C0 01F3           M          addwf   ACCbHI, F       ; add msb
00C1 0333           M NO_ADD   rrf     ACCbHI, F
00C2 0332           M          rrf     ACCbLO, F
00C3 0335           M          rrf     ACCcHI, F
00C4 0334           M          rrf     ACCcLO, F
                    M ;
                  00110        mulMac
  0000              M          LOCAL   NO_ADD
                    M ;
00C5 0337           M          rrf     ACCdHI, F       ; rotate d right
00C6 0336           M          rrf     ACCdLO, F
00C7 0703           M          btfss   STATUS,C        ; need to add?
00C8 0ACF           M          goto    NO_ADD          ; no addition necessary
00C9 0210           M          movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
00CA 01F2           M          addwf   ACCbLO, F       ; add lsb
00CB 0603           M          btfsc   STATUS,C        ; add in carry
00CC 02B3           M          incf    ACCbHI, F
00CD 0211           M          movf    ACCaHI,W
00CE 01F3           M          addwf   ACCbHI, F       ; add msb
00CF 0333           M NO_ADD   rrf     ACCbHI, F
00D0 0332           M          rrf     ACCbLO, F
00D1 0335           M          rrf     ACCcHI, F
00D2 0334           M          rrf     ACCcLO, F
                    M ;
                  00111        mulMac
  0000              M          LOCAL   NO_ADD
                    M ;
00D3 0337           M          rrf     ACCdHI, F       ; rotate d right
00D4 0336           M          rrf     ACCdLO, F
00D5 0703           M          btfss   STATUS,C        ; need to add?
00D6 0ADD           M          goto    NO_ADD          ; no addition necessary
00D7 0210           M          movf    ACCaLO,W        ; Addition ( ACCb + ACCa -> ACCb )
00D8 01F2           M          addwf   ACCbLO, F       ; add lsb
00D9 0603           M          btfsc   STATUS,C        ; add in carry
00DA 02B3           M          incf    ACCbHI, F
00DB 0211           M          movf    ACCaHI,W
00DC 01F3           M          addwf   ACCbHI, F       ; add msb
00DD 0333           M NO_ADD   rrf     ACCbHI, F
00DE 0332           M          rrf     ACCbLO, F
00DF 0335           M          rrf     ACCcHI, F
```


© 1997 Microchip Technology Inc.

```
00E0 0334          M         rrf     ACCcLO, F
                   M ;
                00112 ;
                00113     IF    SIGNED
                00114         btfss   sign,MSB
                00115         retlw   0
                00116         comf    ACCcLO          ; negate ACCa ( -ACCa -> ACCa )
                00117         incf    ACCcLO
                00118         btfsc   STATUS,Z
                00119         decf    ACCcHI
                00120         comf    ACCcHI
                00121         btfsc   STATUS,Z
                00122 neg_B   comf    ACCbLO          ; negate ACCb
                00123         incf    ACCbLO
                00124         btfsc   STATUS,Z
                00125         decf    ACCbHI
                00126         comf    ACCbHI
                00127         retlw   0
                00128     ELSE
00E1 0800       00129         retlw   0
                00130     ENDIF
                00131 ;
                00132 ;*****************************************************************
                00133 ;
00E2 0C10       00134 setup   movlw   .16             ; for 16 shifts
00E3 0038       00135         movwf   temp
00E4 0213       00136         movf    ACCbHI,W          ;move ACCb to ACCd
00E5 0037       00137         movwf   ACCdHI
00E6 0212       00138         movf    ACCbLO,W
00E7 0036       00139         movwf   ACCdLO
00E8 0073       00140         clrf    ACCbHI
00E9 0072       00141         clrf    ACCbLO
00EA 0800       00142         retlw   0
                00143 ;
                00144 ;*****************************************************************
                00145 ;
00EB 0270       00146 neg_A   comf    ACCaLO, F       ; negate ACCa ( -ACCa -> ACCa )
00EC 02B0       00147         incf    ACCaLO, F
00ED 0643       00148         btfsc   STATUS,Z
00EE 00F1       00149         decf    ACCaHI, F
00EF 0271       00150         comf    ACCaHI, F
00F0 0800       00151         retlw   0
                00152 ;
                00153 ;*****************************************************************
                00154 ;  Assemble this section only if Signed Arithmetic Needed
                00155 ;
                00156       IF    SIGNED
                00157 ;
                00158 S_SIGN  movf    ACCaHI,W
                00159         xorwf   ACCbHI,W
                00160         movwf   sign
                00161         btfss   ACCbHI,MSB        ; if MSB set go & negate ACCb
                00162         goto    chek_A
                00163 ;
                00164         comf    ACCbLO          ; negate ACCb
                00165         incf    ACCbLO
                00166         btfsc   STATUS,Z
                00167         decf    ACCbHI
                00168         comf    ACCbHI
                00169 ;
                00170 chek_A  btfss   ACCaHI,MSB        ; if MSB set go & negate ACCa
                00171         retlw   0
                00172         goto    neg_A
                00173 ;
                00174       ENDIF
                00175 ;
```


© 1997 Microchip Technology Inc.

```
                00176 ;*******************************************************************
                00177 ;                         Test Program
                00178 ;*******************************************************************
                00179 ;    Load constant values to ACCa & ACCb for testing
                00180 ;
00F1 0C01       00181 loadAB  movlw   1
00F2 0031       00182         movwf   ACCaHI
00F3 0CFF       00183         movlw   0FF             ; loads ACCa = 01FF
00F4 0030       00184         movwf   ACCaLO
                00185 ;
00F5 0C7F       00186         movlw   07F
00F6 0033       00187         movwf   ACCbHI
00F7 0CFF       00188         movlw   0FF             ; loads ACCb = 7FFF
00F8 0032       00189         movwf   ACCbLO
00F9 0800       00190         retlw   0
                00191 ;
00FA 0000       00192 main    nop
                00193 ;
00FB 09F1       00194         call    loadAB ;result of multiplying ACCb*ACCa->(ACCb,ACCc)
00FC 0900       00195         call    D_mpyF          ; Here (ACCb,ACCc) = 00FF 7E01
                00196 ;
00FD 0AFD       00197 self    goto    self
                00198 ;
01FF            00199         org     PIC54
01FF 0AFA       00200         goto    main
                00201         END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX--
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   255
Program Memory Words Free:   257


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX E:DOUBLE PRECISION ADDITION AND SUBTRACTION LISTING

```
MPASM 01.40 Released          DBL_ADD.ASM   1-16-1997  12:50:38         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;                 Double Precision Addition & Subtraction
                     00005 ;
                     00006 ;*****************************************************************;
                     00007 ;   Addition :  ACCb(16 bits) + ACCa(16 bits) -> ACCb(16 bits)
                     00008 ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                     00009 ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                     00010 ;      (c) CALL D_add
                     00011 ;      (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
                     00012 ;
                     00013 ;   Performance :
                     00014 ;               Program Memory  :  07
                     00015 ;               Clock Cycles    :  08
                     00016 ;*****************************************************************;
                     00017 ;   Subtraction : ACCb(16 bits) - ACCa(16 bits) -> ACCb(16 bits)
                     00018 ;      (a) Load the 1st operand in location ACCaLO & ACCaHI ( 16 bits )
                     00019 ;      (b) Load the 2nd operand in location ACCbLO & ACCbHI ( 16 bits )
                     00020 ;      (c) CALL D_sub
                     00021 ;      (d) The result is in location ACCbLO & ACCbHI ( 16 bits )
                     00022 ;
                     00023 ;   Performance :
                     00024 ;               Program Memory  :       14
                     00025 ;               Clock Cycles    :       17
                     00026 ;
                     00027 ;
                     00028 ;       Program:          DBL_ADD.ASM
                     00029 ;       Revision Date:
                     00030 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00031 ;
                     00032 ;*****************************************************************;
                     00033 ;
  00000010           00034 ACCaLO  equ     10
  00000011           00035 ACCaHI  equ     11
  00000012           00036 ACCbLO  equ     12
  00000013           00037 ACCbHI  equ     13
                     00038 ;
                     00039         include "p16c5x.inc"
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology, Inc.
                     00224         LIST
                     00040
  000001FF           00041 PIC54   equ     1FFH    ; Define Reset Vector
                     00042
0000                 00043         org     0
                     00044 ;*****************************************************************
                     00045 ;         Double Precision Subtraction ( ACCb - ACCa -> ACCb )
                     00046 ;
0000 0908            00047 D_sub   call    neg_A           ; At first negate ACCa; Then add
                     00048 ;
                     00049 ;*****************************************************************
                     00050 ;       Double Precision  Addition ( ACCb + ACCa -> ACCb )
                     00051 ;
0001 0210            00052 D_add   movf    ACCaLO,W
0002 01F2            00053         addwf   ACCbLO, F       ; add lsb
```


© 1997 Microchip Technology Inc.

```
0003 0603      00054          btfsc   STATUS,C        ; add in carry
0004 02B3      00055          incf    ACCbHI, F
0005 0211      00056          movf    ACCaHI,W
0006 01F3      00057          addwf   ACCbHI, F       ; add msb
0007 0800      00058          retlw   0
               00059 ;
               00060 ;
0008 0270      00061 neg_A    comf    ACCaLO, F       ; negate ACCa ( -ACCa -> ACCa )
0009 02B0      00062          incf    ACCaLO, F
000A 0643      00063          btfsc   STATUS,Z
000B 00F1      00064          decf    ACCaHI, F
000C 0271      00065          comf    ACCaHI, F
000D 0800      00066          retlw   0
               00067 ;
               00068 ;*****************************************************************
               00069 ;                         Test Program
               00070 ;*****************************************************************
               00071 ;    Load constant values to ACCa & ACCb for testing
               00072 ;
000E 0C01      00073 loadAB   movlw   1
000F 0031      00074          movwf   ACCaHI
0010 0CFF      00075          movlw   0FF             ; loads ACCa = 01FF
0011 0030      00076          movwf   ACCaLO
               00077 ;
0012 0C7F      00078          movlw   07F
0013 0033      00079          movwf   ACCbHI
0014 0CFF      00080          movlw   0FF             ; loads ACCb = 7FFF
0015 0032      00081          movwf   ACCbLO
0016 0800      00082          retlw   0
               00083 ;
0017 0000      00084 main     nop
               00085 ;
0018 090E      00086          call    loadAB          ; result of adding ACCb+ACCa->ACCb
0019 0901      00087          call    D_add           ; Here Accb = 81FE
               00088 ;
001A 090E      00089          call    loadAB     ; result of subtracting ACCb - ACCa->ACCb
001B 0900      00090          call    D_sub           ; Here Accb = 7E00
               00091 ;
001C 0A1C      00092 self     goto    self
               00093 ;
01FF           00094          org     PIC54
01FF 0A17      00095          goto    main
               00096          END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXX--- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:    30
Program Memory Words Free:   482


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX F:BCD TO BINARY CONVERSION LISTING

```
MPASM 01.40 Released          BCD2BIN.ASM   1-16-1997  12:49:30         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;               BCD To Binary Conversion
                     00005 ;
                     00006 ;       This routine converts a 5 digit BCD number to a 16 bit binary
                     00007 ; number.
                     00008 ;       The input 5 digit BCD numbers are asumed to be in locations
                     00009 ; R0, R1 & R2 with R0 containing the MSD in its right most nibble.
                     00010 ;
                     00011 ;       The 16 bit binary number is output in registers H_byte & L_byte
                     00012 ; ( high byte & low byte repectively ).
                     00013 ;
                     00014 ;       The method used for conversion is :
                     00015 ;               input number X = abcde ( the 5 digit BCD number )
                     00016 ;               X = abcde = 10[10[10[10a+b]+c]+d]+e
                     00017 ;
                     00018 ;   Performance :
                     00019 ;               Program Memory  :       30
                     00020 ;               Clock Cycles    :       121
                     00021 ;
                     00022 ;
                     00023 ;       Program:          BCD2BIN.ASM
                     00024 ;       Revision Date:
                     00025 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00026 ;
                     00027 ;*****************************************************************;
                     00028 ;
  00000010           00029 H_byte  equ     10
  00000011           00030 L_byte  equ     11
  00000012           00031 R0      equ     12              ; RAM Assignments
  00000013           00032 R1      equ     13
  00000014           00033 R2      equ     14
                     00034 ;
  00000015           00035 H_temp  equ     15      ; temporary register
  00000016           00036 L_temp  equ     16      ; temporary register
                     00037 ;
                     00038 ;
                     00039         INCLUDE         "p16c5x.inc"
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology, Inc.
                     00224         LIST
                     00040 ;
                     00041 ;
0000 0E0F            00042 mpy10b  andlw   0F
0001 01F1            00043         addwf   L_byte, F
0002 0603            00044         btfsc   STATUS,C
0003 02B0            00045         incf    H_byte, F
0004 0403            00046 mpy10a  bcf     STATUS,C        ; multiply by 2
0005 0351            00047         rlf     L_byte,W
0006 0036            00048         movwf   L_temp
0007 0350            00049         rlf     H_byte,W        ; (H_temp,L_temp) = 2*N
0008 0035            00050         movwf   H_temp
                     00051 ;
```


© 1997 Microchip Technology Inc.

```
0009 0403           00052          bcf     STATUS,C        ; multiply by 2
000A 0371           00053          rlf     L_byte, F
000B 0370           00054          rlf     H_byte, F
000C 0403           00055          bcf     STATUS,C        ; multiply by 2
000D 0371           00056          rlf     L_byte, F
000E 0370           00057          rlf     H_byte, F
000F 0403           00058          bcf     STATUS,C        ; multiply by 2
0010 0371           00059          rlf     L_byte, F
0011 0370           00060          rlf     H_byte, F       ; (H_byte,L_byte) = 8*N
                    00061 ;
0012 0216           00062          movf    L_temp,W
0013 01F1           00063          addwf   L_byte, F
0014 0603           00064          btfsc   STATUS,C
0015 02B0           00065          incf    H_byte, F
0016 0215           00066          movf    H_temp,W
0017 01F0           00067          addwf   H_byte, F
0018 0800           00068          retlw   0               ; (H_byte,L_byte) = 10*N
                    00069 ;
                    00070 ;
0019 0070           00071 BCDtoB   clrf    H_byte
001A 0212           00072          movf    R0,W
001B 0E0F           00073          andlw   0F
001C 0031           00074          movwf   L_byte
001D 0904           00075          call    mpy10a          ; result = 10a+b
                    00076 ;
001E 0393           00077          swapf   R1,W
001F 0900           00078          call    mpy10b          ; result = 10[10a+b]
                    00079 ;
0020 0213           00080          movf    R1,W
0021 0900           00081          call    mpy10b          ; result = 10[10[10a+b]+c]
                    00082 ;
0022 0394           00083          swapf   R2,W
0023 0900           00084          call    mpy10b          ; result = 10[10[10[10a+b]+c]+d]
                    00085 ;
0024 0214           00086          movf    R2,W
0025 0E0F           00087          andlw   0F
0026 01F1           00088          addwf   L_byte, F
0027 0603           00089          btfsc   STATUS,C
0028 02B0           00090          incf    H_byte, F       ; result = 10[10[10[10a+b]+c]+d]+e
0029 0800           00091          retlw   0               ; BCD to binary conversion done
                    00092 ;
                    00093 ;
                    00094 ;*************************************************************************
                    00095 ;               Test Program
                    00096 ;*************************************************************************
002A 0C06           00097 main     movlw   06
002B 0032           00098          movwf   R0      ; Set R0 = 06
002C 0C55           00099          movlw   55
002D 0033           00100          movwf   R1      ; Set R1 = 55
002E 0C35           00101          movlw   35
002F 0034           00102          movwf   R2      ; Set R2 = 35      ( R0, R1, R2 = 6,55,35 )
                    00103 ;
0030 0919           00104          call    BCDtoB  ; After conversion H_Byte = FF & L_Byte = FF
                    00105 ;
0031 0A31           00106 self     goto    self
                    00107 ;
01FF                00108          org     1FF
01FF 0A2A           00109          goto    main
                    00110 ;
                    00111          END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XX--------------
01C0 : ---------------- ---------------- ---------------- --------------X

All other memory blocks unused.
```

5

```
Program Memory Words Used:     51
Program Memory Words Free:    461

Errors   :      0
Warnings :      0 reported,      0 suppressed
Messages :      0 reported,      0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX G:BINARY (8-BIT) TO BCD CONVERSION

```
MPASM 01.40 Released          BIN8BCD.ASM   1-16-1997  12:50:05         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;                  Binary To BCD Conversion Routine
                     00005 ;
                     00006 ;       This routine converts the 8 bit binary number in the W Register
                     00007 ; to a 2 digit BCD number.
                     00008 ;       The least significant digit is returned in location LSD and
                     00009 ; the most significant digit is returned in location MSD.
                     00010 ;
                     00011 ;   Performance :
                     00012 ;               Program Memory  :       10
                     00013 ;               Clock Cycles    :       81 (worst case when W = 63 Hex )
                     00014 ;                                          ( i.e max Decimal number 99 )
                     00015 ;
                     00016 ;       Program:          BIN8BCD.ASM
                     00017 ;       Revision Date:
                     00018 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00019 ;
                     00020 ;*****************************************************************
                     00021 ;
  00000010           00022 LSD     equ     10
  00000011           00023 MSD     equ     11
                     00024 ;
                     00025         INCLUDE         "p16c5x.inc"
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology, Inc.
                     00224         LIST
                     00026 ;
0000 0071            00027 BinBCD  clrf    MSD
0001 0030            00028         movwf   LSD
0002 0C0A            00029 gtenth  movlw   .10
0003 0090            00030         subwf   LSD,W
0004 0703            00031         BTFSS   STATUS,C
0005 0A09            00032         goto    over
0006 0030            00033         movwf   LSD
0007 02B1            00034         incf    MSD, F
0008 0A02            00035         goto    gtenth
0009 0800            00036 over    retlw   0
                     00037 ;*****************************************************************
                     00038 ;
000A 0C63            00039 main    movlw   63              ; W reg = 63 Hex
000B 0900            00040         call    BinBCD          ; after conversion, MSD = 9 & LSD = 9
000C 0A0C            00041 self    goto    self            ; ( 63 Hex = 99 Decimal )
                     00042 ;
01FF                 00043         org     1FF
01FF 0A0A            00044         goto    main
                     00045 ;
                     00046         END
0000 : XXXXXXXXXXXXX--- ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.
```

```
Program Memory Words Used:     14
Program Memory Words Free:    498


Errors   :      0
Warnings :      0 reported,      0 suppressed
Messages :      0 reported,      0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX H:BINARY (16-BIT) TO BCD LISTING

```
MPASM 01.40 Released          B16TOBCD.ASM   1-16-1997  12:48:00         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;                  Binary To BCD Conversion Routine
                     00005 ;       This routine converts a 16 Bit binary Number to a 5 Digit
                     00006 ; BCD Number. This routine is useful since PIC16C55 & PIC16C57
                     00007 ; have  two 8 bit ports and one 4 bit port ( total of 5 BCD digits)
                     00008 ;
                     00009 ;       The 16 bit binary number is input in locations H_byte and
                     00010 ; L_byte with the high byte in H_byte.
                     00011 ;       The 5 digit BCD number is returned in R0, R1 and R2 with R0
                     00012 ; containing the MSD in its right most nibble.
                     00013 ;
                     00014 ;   Performance :
                     00015 ;               Program Memory  :       35
                     00016 ;               Clock Cycles    :       885
                     00017 ;
                     00018 ;
                     00019 ;       Program:          B16TOBCD.ASM
                     00020 ;       Revision Date:
                     00021 ;                         1-13-97    Compatibility with MPASMWIN 1.40
                     00022 ;
                     00023 ;*****************************************************************;
                     00024 ;
  00000016           00025 count  equ     16
  00000017           00026 temp   equ     17
                     00027 ;
  00000010           00028 H_byte  equ    10
  00000011           00029 L_byte  equ    11
  00000012           00030 R0      equ    12              ; RAM Assignments
  00000013           00031 R1      equ    13
  00000014           00032 R2      equ    14
                     00033 ;
                     00034         include         "p16c5x.inc"
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology, Inc.
                     00224         LIST
                     00035 ;
0000 0403            00036 B2_BCD  bcf     STATUS,0                ; clear the carry bit
0001 0C10            00037         movlw   .16
0002 0036            00038         movwf   count
0003 0072            00039         clrf    R0
0004 0073            00040         clrf    R1
0005 0074            00041         clrf    R2
0006 0371            00042 loop16  rlf     L_byte, F
0007 0370            00043         rlf     H_byte, F
0008 0374            00044         rlf     R2, F
0009 0373            00045         rlf     R1, F
000A 0372            00046         rlf     R0, F
                     00047 ;
000B 02F6            00048         decfsz  count, F
000C 0A0E            00049         goto    adjDEC
000D 0800            00050         RETLW   0
                     00051 ;
```

```
000E 0C14       00052 adjDEC  movlw   R2
000F 0024       00053         movwf   FSR
0010 0918       00054         call    adjBCD
                00055 ;
0011 0C13       00056         movlw   R1
0012 0024       00057         movwf   FSR
0013 0918       00058         call    adjBCD
                00059 ;
0014 0C12       00060         movlw   R0
0015 0024       00061         movwf   FSR
0016 0918       00062         call    adjBCD
                00063 ;
0017 0A06       00064         goto    loop16
                00065 ;
0018 0C03       00066 adjBCD  movlw   3
0019 01C0       00067         addwf   0,W
001A 0037       00068         movwf   temp
001B 0677       00069         btfsc   temp,3          ; test if result > 7
001C 0020       00070         movwf   0
001D 0C30       00071         movlw   30
001E 01C0       00072         addwf   0,W
001F 0037       00073         movwf   temp
0020 06F7       00074         btfsc   temp,7          ; test if result > 7
0021 0020       00075         movwf   0               ; save as MSD
0022 0800       00076         RETLW   0
                00077 ;
                00078 ;*****************************************************************
                00079 ;               Test Program
                00080 ;*****************************************************************
0023 0CFF       00081 main    movlw   0FF
0024 0030       00082         movwf   H_byte
0025 0031       00083         movwf   L_byte          ; The 16 bit binary number = FFFF
0026 0900       00084         call    B2_BCD          ; After conversion the Decimal Number
                00085 ;                               ; in R0,R1,R2 = 06,55,35
                00086 ;
0027 0A27       00087 self    goto    self
                00088 ;
01FF            00089         org     1FF
01FF 0A23       00090         goto    main
                00091 ;
                00092         END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXX-------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:    41
Program Memory Words Free:   471

Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX I:UNSIGNED BCD SUBTRACTION LISTING

```
MPASM 01.40 Released          BCD_SUB.ASM   1-16-1997  12:49:00         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;******************  Unsigned BCD Subtraction   ***************
                     00004 ;
                     00005 ;       This routine performs a 2 Digit Unsigned BCD Subtraction.
                     00006 ; It is assumed that the two BCD numbers to be subtracted are in
                     00007 ; locations Num_1 & Num_2. The result is the difference of Num_1 & Num_2
                     00008 ; ( Num_2 - Num_1) and is stored in location Num_2 and the overflow carry
                     00009 ; is returned in location Num_1.
                     00010 ;
                     00011 ;   Performance :
                     00012 ;               Program Memory  :       31
                     00013 ;               Clock Cycles    :       21   ( worst case )
                     00014 ;
                     00015 ;
                     00016 ;       Program:          BCD_SUB.ASM
                     00017 ;       Revision Date:
                     00018 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00019 ;
                     00020 ;*************************************************************************
                     00021 ;
  00000008           00022 Num_1   equ     8       ; Overflow flow carry overwrites Num_1
  00000008           00023 result  equ     8
                     00024 ;
  00000009           00025 Num_2   equ     9       ; Num_2 - Num_1 overwrites Num_2
  00000009           00026 O_flow  equ     9
                     00027 ;
                     00028         include         "p16c5x.inc"
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology, Inc.
                     00224         LIST
                     00029 ;
0000 0208            00030 BCDSub  movf    Num_1,W
0001 00A9            00031         subwf   Num_2, F
0002 0068            00032         clrf    Num_1
0003 0368            00033         rlf     Num_1, F
0004 0723            00034         btfss   STATUS,DC
0005 0A0C            00035         goto    adjst1
0006 0769            00036         btfss   Num_2,3         ; Adjust LSD of Result
0007 0A0E            00037         goto    Over_1
0008 0649            00038         btfsc   Num_2,2
0009 0A0C            00039         goto    adjst1          ; Adjust LSD of Result
000A 0729            00040         btfss   Num_2,1
000B 0A0E            00041         goto    Over_1          ; No : Go for MSD
000C 0C06            00042 adjst1  movlw   6
000D 00A9            00043         subwf   Num_2, F
000E 0708            00044 Over_1  btfss   Num_1,0         ; CY = 0 ?
000F 0A17            00045         goto    adjst2          ; Yes, adjust MSD of result
0010 0068            00046         clrf    Num_1
0011 07E9            00047         btfss   Num_2,7         ; No, test for MSD >9
0012 0800            00048         RETLW   0
0013 06C9            00049         btfsc   Num_2,6
0014 0A17            00050         goto    adjst2
0015 07A9            00051         btfss   Num_2,5
```

```
0016 0800       00052           RETLW   0
0017 0C60       00053 adjst2    movlw   60              ; add 6 to MSD
0018 00A9       00054           subwf   Num_2, F
0019 0068       00055           clrf    Num_1
001A 0703       00056           btfss   STATUS,C        ; test if underflow
001B 0800       00057           RETLW   0
001C 0C01       00058           movlw   1
001D 0028       00059           movwf   Num_1
001E 0800       00060 Over      RETLW   0
                00061 ;
                00062 ;*****************************************************************
                00063 ;                 Test Program
                00064 ;*****************************************************************
001F 0C23       00065 main      movlw   23
0020 0028       00066           movwf   Num_1       ; Set Num_1 = 23
0021 0C99       00067           movlw   99
0022 0029       00068           movwf   Num_2       ; Set Num_2 = 99
0023 0900       00069           call    BCDSub      ; After subtraction, Num_2 = 76 ( 99-23 )
                00070 ;                             ;  and Num_1 = 0 ( indicates positive result )
                00071 ;
0024 0C99       00072           movlw   99
0025 0028       00073           movwf   Num_1       ; Set Num_1 = 99
0026 0C00       00074           movlw   0
0027 0029       00075           movwf   Num_2       ; Set Num_2 = 0
                00076 ;
0028 0900       00077           call    BCDSub      ; After subtraction, Num_2 = 1
                00078 ;                             ;  and Num_1 = 1 ( indicates negative result )
                00079 ;                             ;  -1  <-  ( -99 )
                00080 ;
0029 0A29       00081 self      goto    self
                00082 ;
01FF            00083           org     1FF
01FF 0A1F       00084           goto    main
                00085 ;
                00086           END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXX------ ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:    43
Program Memory Words Free:   469

Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX J:SQUARE ROOT METHOD

```
MPASM 01.40 Released              SQRT.ASM   1-16-1997  12:55:13          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;
                     00005 ;             Square Root By Newton Raphson Method
                     00006 ;
                     00007 ;  This routine computes the square root of a 16 bit number(with
                     00008 ;  low byte in NumLo & high byte in NumHi ). After loading NumLo &
                     00009 ;  NumHi with the desired number whose square root is to be computed,
                     00010 ;  branch to location Sqrt ( by "GOTO  Sqrt" ). " CALL  Sqrt" cannot
                     00011 ;  be issued because the Sqrt function makes calls to Math routines
                     00012 ;  and the stack is completely used up.
                     00013 ;  The result = sqrt(NumHi,NumLo) is returned in location SqrtLo.
                     00014 ;  The total number of iterations is set to ten. If more iterations
                     00015 ;  are desired, change "LupCnt equ .10" to the desired value. Also,
                     00016 ;  the initial guess value of the square root is given set as
                     00017 ;  input/2 ( in subroutine "init" ). The user may modify this scheme
                     00018 ;  if a better initial approximation value is known. A good initial
                     00019 ;  guess will help the algorithm converge at a faster rate and thus
                     00020 ;  less number of iterations required.
                     00021 ;  Two utility math routines are used by this program : D_divS
                     00022 ;  and D_add. These two routines are listed as seperate routines
                     00023 ;  under double precision Division and double precision addtion
                     00024 ;  respectively.
                     00025 ;
                     00026 ;  Note : If square root of an 8 bit number is desired, it is probably
                     00027 ;         better to have a table look scheme rather than using numerical
                     00028 ;         methods.
                     00029 ;
                     00030 ;
                     00031 ;
                     00032 ;   Performance :
                     00033 ;                Program Memory  :       27  (excluding Math Routines
                     00034 ;                                            D_divS & D_add )
                     00035 ;                Clock Cycles    :       3600 ( approximately )
                     00036 ;
                     00037 ;
                     00038 ;        Program:          SQRT.ASM
                     00039 ;        Revision Date:
                     00040 ;                          1-13-97      Compatibility with MPASMWIN 1.40
                     00041 ;
                     00042 ;        To assemble this program, two routines, namely "D_add" &
                     00043 ;        "D_divS" must be included into this program. These two routines
                     00044 ;        are listed as separate programs in files "DBL_ADD.ASM" &
                     00045 ;        "DBL_DIVS.ASM" respectively.
                     00046 ;
                     00047 ;*****************************************************************
                     00048         include "p16c5x.inc"
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File, Ver.n 3.30 Microchip Technology, Inc.
                     00224         LIST
                     00049
  000001FF           00050 PIC54   equ     1FFH    ; Define Reset Vector
  00000001           00051 TRUE    equ     1
```


© 1997 Microchip Technology Inc.

```
  00000000      00052 FALSE   equ     0
                00053
0000            00054         org     0
                00055 ;
  0000000A      00056 LupCnt  equ     .10             ; Number of iterations
                00057 ;
  00000010      00058 ACCaLO  equ     10
  00000011      00059 ACCaHI  equ     11
  00000013      00060 ACCbLO  equ     13
  00000014      00061 ACCbHI  equ     14
  00000014      00062 ACCcLO  equ     14
  00000015      00063 ACCcHI  equ     15
  00000016      00064 ACCdLO  equ     16
  00000017      00065 ACCdHI  equ     17
  00000018      00066 temp    equ     18
  00000019      00067 sign    equ     19
                00068 ;
  00000010      00069 SqrtLo  equ     ACCaLO
  00000011      00070 SqrtHi  equ     ACCaHI
                00071 ;
  0000001D      00072 NumLo   equ     1D
  0000001E      00073 NumHi   equ     1E
  0000001F      00074 count   equ     1F
                00075 ;
                00076 ;
0000            00077 init
0000 0C0A       00078         movlw   LupCnt
0001 003F       00079         movwf   count
0002 021E       00080         movf    NumHi,W
0003 0031       00081         movwf   SqrtHi
0004 021D       00082         movf    NumLo,W         ; set initial guess root = NUM/2
0005 0030       00083         movwf   SqrtLo
0006 0403       00084         bcf     STATUS,C
0007 0331       00085         rrf     SqrtHi, F
0008 0330       00086         rrf     SqrtLo, F
0009 0800       00087         retlw   0
                00088 ;
000A 0403       00089 div2    bcf     STATUS,C
000B 0314       00090         rrf     ACCbHI,W
000C 0031       00091         movwf   SqrtHi
000D 0313       00092         rrf     ACCbLO,W
000E 0030       00093         movwf   SqrtLo
000F 0800       00094         retlw   0
                00095 ;
                00096 ;*****************************************************************
                00097 ;       Double Precision  Addition ( ACCb + ACCa -> ACCb )
                00098 ;
0010 0210       00099 D_add   movf    ACCaLO,W
0011 01F3       00100         addwf   ACCbLO, F       ;add lsb
0012 0603       00101         btfsc   STATUS,C    ;add in carry
0013 02B4       00102         incf    ACCbHI, F
0014 0211       00103         movf    ACCaHI,W
0015 01F4       00104         addwf   ACCbHI, F       ;add msb
0016 0800       00105         retlw   0
                00106 ;
                00107 ;*****************************************************************
  00000000      00108 SIGNED  equ     FALSE           ; Set This To 'TRUE' if the routines
                00109 ;                               ; for Multiplication & Division needs
                00110 ;                               ; to be assembled as Signed Integer
                00111 ;                               ; Routines. If 'FALSE' the above two
                00112 ;                               ; routines ( D_mpy & D_div ) use
                00113 ;                               ; unsigned arithmetic.
                00114 ;*****************************************************************
                00115 ;       Double Precision Divide ( 16/16 -> 16 )
                00116 ;
                00117 ;         ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
```


© 1997 Microchip Technology Inc.

```
                00118 ; with Quotiont in ACCb (ACCbHI,ACCbLO) and Remainder in
                00119 ; ACCc (ACCcHI,ACCcLO).
                00120 ;   NOTE: Before calling this routine, the user should make sure that
                00121 ;         the Numerator(ACCb) is greater than Denominator(ACCa). If
                00122 ;         the case is not true, the user should scale either Numerator
                00123 ;         or Denominator or both such that Numerator is greater than
                00124 ;         the Denominator.
                00125 ;
                00126 ;
0017            00127 D_divS
                00128 ;
                00129       IF   SIGNED
                00130       CALL    S_SIGN
                00131       ENDIF
                00132 ;
0017 0933       00133          call    setup
0018 0075       00134          clrf    ACCcHI
0019 0074       00135          clrf    ACCcLO
001A 0403       00136 dloop    bcf     STATUS,C
001B 0376       00137          rlf     ACCdLO, F
001C 0377       00138          rlf     ACCdHI, F
001D 0374       00139          rlf     ACCcLO, F
001E 0375       00140          rlf     ACCcHI, F
001F 0211       00141          movf    ACCaHI,W
0020 0095       00142          subwf   ACCcHI,W          ; check if a>c
0021 0743       00143          btfss   STATUS,Z
0022 0A25       00144          goto    nochk
0023 0210       00145          movf    ACCaLO,W
0024 0094       00146          subwf   ACCcLO,W          ; if msb equal then check lsb
0025 0703       00147 nochk    btfss   STATUS,C          ; carry set if c>a
0026 0A2E       00148          goto    nogo
0027 0210       00149          movf    ACCaLO,W          ; c-a into c
0028 00B4       00150          subwf   ACCcLO, F
0029 0703       00151          btfss   STATUS,C
002A 00F5       00152          decf    ACCcHI, F
002B 0211       00153          movf    ACCaHI,W
002C 00B5       00154          subwf   ACCcHI, F
002D 0503       00155          bsf     STATUS,C          ; shift a 1 into b (result)
002E 0373       00156 nogo     rlf     ACCbLO, F
002F 0374       00157          rlf     ACCbHI, F
0030 02F8       00158          decfsz  temp, F           ; loop untill all bits checked
0031 0A1A       00159          goto    dloop
                00160 ;
                00161       IF    SIGNED
                00162          btfss   sign,MSB        ; check sign if negative
                00163          retlw   0
                00164          goto    neg_B           ; negate ACCa ( -ACCa -> ACCa )
                00165       ELSE
0032 0800       00166          retlw   0
                00167       ENDIF
                00168 ;
                00169 ;*****************************************************************
                00170 ;
0033 0C10       00171 setup    movlw   .16             ; for 16 shifts
0034 0038       00172          movwf   temp
0035 0214       00173          movf    ACCbHI,W        ; move ACCb to ACCd
0036 0037       00174          movwf   ACCdHI
0037 0213       00175          movf    ACCbLO,W
0038 0036       00176          movwf   ACCdLO
0039 0074       00177          clrf    ACCbHI
003A 0073       00178          clrf    ACCbLO
003B 0800       00179          retlw   0
                00180 ;
                00181 ;*****************************************************************
                00182 ;
003C 0270       00183 neg_A    comf    ACCaLO, F       ; negate ACCa ( -ACCa -> ACCa )
```


© 1997 Microchip Technology Inc.

```
003D 02B0       00184         incf    ACCaLO, F
003E 0643       00185         btfsc   STATUS,Z
003F 00F1       00186         decf    ACCaHI, F
0040 0271       00187         comf    ACCaHI, F
0041 0800       00188         retlw   0
                00189 ;
                00190 ;*****************************************************************
                00191 ;  Assemble this section only if Signed Arithmetic Needed
                00192 ;
                00193      IF    SIGNED
                00194 ;
                00195 S_SIGN  movf    ACCaHI,W
                00196         xorwf   ACCbHI,W
                00197         movwf   sign
                00198         btfss   ACCbHI,MSB        ; if MSB set go & negate ACCb
                00199         goto    chek_A
                00200 ;
                00201         comf    ACCbLO          ; negate ACCb
                00202         incf    ACCbLO
                00203         btfsc   STATUS,Z
                00204         decf    ACCbHI
                00205         comf    ACCbHI
                00206 ;
                00207 chek_A  btfss   ACCaHI,MSB        ; if MSB set go & negate ACCa
                00208         retlw   0
                00209         goto    neg_A
                00210 ;
                00211      ENDIF
                00212 ;
                00213 ;
0042 0900       00214 Sqrt    call    init
0043 021D       00215 sloop   movf    NumLo,W
0044 0033       00216         movwf   ACCbLO
0045 021E       00217         movf    NumHi,W
0046 0034       00218         movwf   ACCbHI
                00219 ;
0047 0917       00220         call    D_divS          ; double precision division
0048 0910       00221         call    D_add           ; double precision addition
                00222 ;                               ; the above 2 routines are listed
                00223 ;                               ; as seperate routines
0049 090A       00224         call    div2
004A 02FF       00225         decfsz  count, F
004B 0A43       00226         goto    sloop
004C 0A52       00227         goto    over            ; all iterations done
                00228 ;                               ; branch back to desired location
                00229 ;
                00230 ;*************************************************************
                00231 ;               Test Program
                00232 ;*************************************************************
                00233 ;
004D 0CF3       00234 main    movlw   0F3
004E 003E       00235         movwf   NumHi
004F 0CF6       00236         movlw   0F6       ; Set input test number = 62454
0050 003D       00237         movwf   NumLo     ;  = F3F6h
                00238 ;
0051 0A42       00239         goto    Sqrt      ; cannot use CALL : Math routines
                00240 ;                         ; use up all the stack.
0052 0000       00241 over    nop               ; all iterations done
                00242 ;
0053 0A53       00243 self    goto    self      ; result = 00F9h = 249
                00244 ;                         ; exact sqrt(62454) = 249.9
                00245 ;
01FF            00246         org     PIC54
01FF 0A4D       00247         goto    main
                00248 ;
                00249         END
```


© 1997 Microchip Technology Inc.

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXX------------ ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:    85
Program Memory Words Free:   427


Errors   : 0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX K:DOUBLE PRECISION DIVISION LISTING (LOOPED)

```
MPASM 01.40 Released         DBL_DIVF.ASM   1-16-1997  12:51:16         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;                       Double Precision Division
                     00005 ;
                     00006 ;               ( Optimized for Speed : straight Line Code )
                     00007 ;
                     00008 ;*****************************************************************;
                     00009 ; Division : ACCb(16 bits)/ACCa(16 bits)-> ACCb(16 bits) with
                     00010 ;                                          Remainder in ACCc (16 bits)
                     00011 ;       (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
                     00012 ;       (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
                     00013 ;       (c) CALL D_div
                     00014 ;       (d) The 16 bit result is in location ACCbHI & ACCbLO
                     00015 ;       (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
                     00016 ;
                     00017 ;   Performance :
                     00018 ;               Program Memory  :       370
                     00019 ;               Clock Cycles    :       263
                     00020 ;
                     00021 ;       NOTE :
                     00022 ;               The performance specs are for Unsigned arithmetic (i.e,
                     00023 ;               with "SIGNED equ  FALSE").
                     00024 ;
                     00025 ;
                     00026 ;       Program:          DBL_DIVF.ASM
                     00027 ;       Revision Date:
                     00028 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00029 ;
                     00030 ;*****************************************************************;
                     00031 ;
  00000010           00032 ACCaLO  equ     10
  00000011           00033 ACCaHI  equ     11
  00000012           00034 ACCbLO  equ     12
  00000013           00035 ACCbHI  equ     13
  00000014           00036 ACCcLO  equ     14
  00000015           00037 ACCcHI  equ     15
  00000016           00038 ACCdLO  equ     16
  00000017           00039 ACCdHI  equ     17
  00000018           00040 temp    equ     18
  00000019           00041 sign    equ     19
                     00042 ;
                     00043         include "p16c5x.inc"
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology, Inc.
                     00224         LIST
                     00044
  000001FF           00045 PIC54   equ     1FFH    ; Define Reset Vector
  00000001           00046 TRUE    equ     1
  00000000           00047 FALSE   equ     0
                     00048
0000                 00049         org     0
                     00050 ;*****************************************************************
  00000000           00051 SIGNED  equ     FALSE           ; Set This To 'TRUE' if the routines
```


© 1997 Microchip Technology Inc.

```
                00052 ;                                 ; for Multiplication & Division needs
                00053 ;                                 ; to be assembled as Signed Integer
                00054 ;                                 ; Routines. If 'FALSE' the above two
                00055 ;                                 ; routines ( D_mpy & D_div ) use
                00056 ;                                 ; unsigned arithmetic.
                00057 ;*****************************************************************;
                00058 ;       division macro
                00059 ;
                00060 divMac  MACRO
                00061         LOCAL   NOCHK
                00062         LOCAL   NOGO
                00063 ;
                00064         bcf     STATUS,C
                00065         rlf     ACCdLO, F
                00066         rlf     ACCdHI, F
                00067         rlf     ACCcLO, F
                00068         rlf     ACCcHI, F
                00069         movf    ACCaHI,W
                00070         subwf   ACCcHI,W          ; check if a>c
                00071         btfss   STATUS,Z
                00072         goto    NOCHK
                00073         movf    ACCaLO,W
                00074         subwf   ACCcLO,W          ; if msb equal then check lsb
                00075 NOCHK   btfss   STATUS,C          ; carry set if c>a
                00076         goto    NOGO
                00077         movf    ACCaLO,W          ; c-a into c
                00078         subwf   ACCcLO, F
                00079         btfss   STATUS,C
                00080         decf    ACCcHI, F
                00081         movf    ACCaHI,W
                00082         subwf   ACCcHI, F
                00083         bsf     STATUS,C          ; shift a 1 into b (result)
                00084 NOGO    rlf     ACCbLO, F
                00085         rlf     ACCbHI, F
                00086 ;
                00087         ENDM
                00088 ;
                00089 ;*****************************************************************
                00090 ;       Double Precision Divide ( 16/16 -> 16 )
                00091 ;
                00092 ;         ( ACCb/ACCa -> ACCb with remainder in ACCc ) : 16 bit output
                00093 ; with Quotiont in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc
                00094 ; (ACCcHI,ACCcLO).
                00095 ;   NOTE: Before calling this routine, the user should make sure that
                00096 ;         the Numerator(ACCb) is greater than Denominator(ACCa). If
                00097 ;         the case is not true, the user should scale either Numerator
                00098 ;         or Denominator or both such that Numerator is greater than
                00099 ;         the Denominator.
                00100 ;
                00101 ;
0000 0C10       00102 setup   movlw   .16               ; for 16 shifts
0001 0038       00103         movwf   temp
0002 0213       00104         movf    ACCbHI,W          ; move ACCb to ACCd
0003 0037       00105         movwf   ACCdHI
0004 0212       00106         movf    ACCbLO,W
0005 0036       00107         movwf   ACCdLO
0006 0073       00108         clrf    ACCbHI
0007 0072       00109         clrf    ACCbLO
0008 0800       00110         retlw   0
                00111 ;
                00112 ;*****************************************************************
                00113 ;
0009 0270       00114 neg_A   comf    ACCaLO, F           ; negate ACCa ( -ACCa -> ACCa )
000A 02B0       00115         incf    ACCaLO, F
000B 0643       00116         btfsc   STATUS,Z
000C 00F1       00117         decf    ACCaHI, F
```


© 1997 Microchip Technology Inc.

```
000D 0271       00118          comf    ACCaHI, F
000E 0800       00119          retlw   0
                00120 ;
                00121 ;*****************************************************************
                00122
                00123 ;
000F            00124 D_divF
                00125 ;
                00126       IF   SIGNED
                00127       CALL    S_SIGN
                00128       ENDIF
                00129 ;
000F 0900       00130          call    setup
0010 0075       00131          clrf    ACCcHI
0011 0074       00132          clrf    ACCcLO
                00133 ;
                00134 ; use the mulMac macro 16 times
                00135 ;
                00136          divMac
  0000              M          LOCAL   NOCHK
  0000              M          LOCAL   NOGO
                    M ;
0012 0403           M          bcf     STATUS,C
0013 0376           M          rlf     ACCdLO, F
0014 0377           M          rlf     ACCdHI, F
0015 0374           M          rlf     ACCcLO, F
0016 0375           M          rlf     ACCcHI, F
0017 0211           M          movf    ACCaHI,W
0018 0095           M          subwf   ACCcHI,W          ; check if a>c
0019 0743           M          btfss   STATUS,Z
001A 0A1D           M          goto    NOCHK
001B 0210           M          movf    ACCaLO,W
001C 0094           M          subwf   ACCcLO,W          ; if msb equal then check lsb
001D 0703           M NOCHK    btfss   STATUS,C          ; carry set if c>a
001E 0A26           M          goto    NOGO
001F 0210           M          movf    ACCaLO,W          ; c-a into c
0020 00B4           M          subwf   ACCcLO, F
0021 0703           M          btfss   STATUS,C
0022 00F5           M          decf    ACCcHI, F
0023 0211           M          movf    ACCaHI,W
0024 00B5           M          subwf   ACCcHI, F
0025 0503           M          bsf     STATUS,C          ; shift a 1 into b (result)
0026 0372           M NOGO     rlf     ACCbLO, F
0027 0373           M          rlf     ACCbHI, F
                    M ;
                00137          divMac
  0000              M          LOCAL   NOCHK
  0000              M          LOCAL   NOGO
                    M ;
0028 0403           M          bcf     STATUS,C
0029 0376           M          rlf     ACCdLO, F
002A 0377           M          rlf     ACCdHI, F
002B 0374           M          rlf     ACCcLO, F
002C 0375           M          rlf     ACCcHI, F
002D 0211           M          movf    ACCaHI,W
002E 0095           M          subwf   ACCcHI,W          ; check if a>c
002F 0743           M          btfss   STATUS,Z
0030 0A33           M          goto    NOCHK
0031 0210           M          movf    ACCaLO,W
0032 0094           M          subwf   ACCcLO,W          ; if msb equal then check lsb
0033 0703           M NOCHK    btfss   STATUS,C          ; carry set if c>a
0034 0A3C           M          goto    NOGO
0035 0210           M          movf    ACCaLO,W          ; c-a into c
0036 00B4           M          subwf   ACCcLO, F
0037 0703           M          btfss   STATUS,C
0038 00F5           M          decf    ACCcHI, F
```


© 1997 Microchip Technology Inc.

```
0039 0211          M          movf    ACCaHI,W
003A 00B5          M          subwf   ACCcHI, F
003B 0503          M          bsf     STATUS,C        ; shift a 1 into b (result)
003C 0372          M NOGO     rlf     ACCbLO, F
003D 0373          M          rlf     ACCbHI, F
                   M ;
                 00138        divMac
  0000             M          LOCAL   NOCHK
  0000             M          LOCAL   NOGO
                   M ;
003E 0403          M          bcf     STATUS,C
003F 0376          M          rlf     ACCdLO, F
0040 0377          M          rlf     ACCdHI, F
0041 0374          M          rlf     ACCcLO, F
0042 0375          M          rlf     ACCcHI, F
0043 0211          M          movf    ACCaHI,W
0044 0095          M          subwf   ACCcHI,W        ; check if a>c
0045 0743          M          btfss   STATUS,Z
0046 0A49          M          goto    NOCHK
0047 0210          M          movf    ACCaLO,W
0048 0094          M          subwf   ACCcLO,W        ; if msb equal then check lsb
0049 0703          M NOCHK    btfss   STATUS,C        ; carry set if c>a
004A 0A52          M          goto    NOGO
004B 0210          M          movf    ACCaLO,W        ; c-a into c
004C 00B4          M          subwf   ACCcLO, F
004D 0703          M          btfss   STATUS,C
004E 00F5          M          decf    ACCcHI, F
004F 0211          M          movf    ACCaHI,W
0050 00B5          M          subwf   ACCcHI, F
0051 0503          M          bsf     STATUS,C        ; shift a 1 into b (result)
0052 0372          M NOGO     rlf     ACCbLO, F
0053 0373          M          rlf     ACCbHI, F
                   M ;
                 00139        divMac
  0000             M          LOCAL   NOCHK
  0000             M          LOCAL   NOGO
                   M ;
0054 0403          M          bcf     STATUS,C
0055 0376          M          rlf     ACCdLO, F
0056 0377          M          rlf     ACCdHI, F
0057 0374          M          rlf     ACCcLO, F
0058 0375          M          rlf     ACCcHI, F
0059 0211          M          movf    ACCaHI,W
005A 0095          M          subwf   ACCcHI,W          ; check if a>c
005B 0743          M          btfss   STATUS,Z
005C 0A5F          M          goto    NOCHK
005D 0210          M          movf    ACCaLO,W
005E 0094          M          subwf   ACCcLO,W         ; if msb equal then check lsb
005F 0703          M NOCHK    btfss   STATUS,C         ; carry set if c>a
0060 0A68          M          goto    NOGO
0061 0210          M          movf    ACCaLO,W         ; c-a into c
0062 00B4          M          subwf   ACCcLO, F
0063 0703          M          btfss   STATUS,C
0064 00F5          M          decf    ACCcHI, F
0065 0211          M          movf    ACCaHI,W
0066 00B5          M          subwf   ACCcHI, F
0067 0503          M          bsf     STATUS,C         ; shift a 1 into b (result)
0068 0372          M NOGO     rlf     ACCbLO, F
0069 0373          M          rlf     ACCbHI, F
                   M ;
                 00140        divMac
  0000             M          LOCAL   NOCHK
  0000             M          LOCAL   NOGO
                   M ;
006A 0403          M          bcf     STATUS,C
006B 0376          M          rlf     ACCdLO, F
```

5

```
006C 0377          M          rlf     ACCdHI, F
006D 0374          M          rlf     ACCcLO, F
006E 0375          M          rlf     ACCcHI, F
006F 0211          M          movf    ACCaHI,W
0070 0095          M          subwf   ACCcHI,W          ; check if a>c
0071 0743          M          btfss   STATUS,Z
0072 0A75          M          goto    NOCHK
0073 0210          M          movf    ACCaLO,W
0074 0094          M          subwf   ACCcLO,W          ; if msb equal then check lsb
0075 0703          M NOCHK    btfss   STATUS,C          ; carry set if c>a
0076 0A7E          M          goto    NOGO
0077 0210          M          movf    ACCaLO,W          ; c-a into c
0078 00B4          M          subwf   ACCcLO, F
0079 0703          M          btfss   STATUS,C
007A 00F5          M          decf    ACCcHI, F
007B 0211          M          movf    ACCaHI,W
007C 00B5          M          subwf   ACCcHI, F
007D 0503          M          bsf     STATUS,C          ; shift a 1 into b (result)
007E 0372          M NOGO     rlf     ACCbLO, F
007F 0373          M          rlf     ACCbHI, F
                   M ;
                 00141        divMac
  0000             M          LOCAL   NOCHK
  0000             M          LOCAL   NOGO
                   M ;
0080 0403          M          bcf     STATUS,C
0081 0376          M          rlf     ACCdLO, F
0082 0377          M          rlf     ACCdHI, F
0083 0374          M          rlf     ACCcLO, F
0084 0375          M          rlf     ACCcHI, F
0085 0211          M          movf    ACCaHI,W
0086 0095          M          subwf   ACCcHI,W          ; check if a>c
0087 0743          M          btfss   STATUS,Z
0088 0A8B          M          goto    NOCHK
0089 0210          M          movf    ACCaLO,W
008A 0094          M          subwf   ACCcLO,W          ; if msb equal then check lsb
008B 0703          M NOCHK    btfss   STATUS,C          ; carry set if c>a
008C 0A94          M          goto    NOGO
008D 0210          M          movf    ACCaLO,W           ;c-a into c
008E 00B4          M          subwf   ACCcLO, F
008F 0703          M          btfss   STATUS,C
0090 00F5          M          decf    ACCcHI, F
0091 0211          M          movf    ACCaHI,W
0092 00B5          M          subwf   ACCcHI, F
0093 0503          M          bsf     STATUS,C          ; shift a 1 into b (result)
0094 0372          M NOGO     rlf     ACCbLO, F
0095 0373          M          rlf     ACCbHI, F
                   M ;
                 00142        divMac
  0000             M          LOCAL   NOCHK
  0000             M          LOCAL   NOGO
                   M ;
0096 0403          M          bcf     STATUS,C
0097 0376          M          rlf     ACCdLO, F
0098 0377          M          rlf     ACCdHI, F
0099 0374          M          rlf     ACCcLO, F
009A 0375          M          rlf     ACCcHI, F
009B 0211          M          movf    ACCaHI,W
009C 0095          M          subwf   ACCcHI,W          ; check if a>c
009D 0743          M          btfss   STATUS,Z
009E 0AA1          M          goto    NOCHK
009F 0210          M          movf    ACCaLO,W
00A0 0094          M          subwf   ACCcLO,W          ; if msb equal then check lsb
00A1 0703          M NOCHK    btfss   STATUS,C          ; carry set if c>a
00A2 0AAA          M          goto    NOGO
00A3 0210          M          movf    ACCaLO,W          ; c-a into c
```


© 1997 Microchip Technology Inc.

```
00A4 00B4          M          subwf   ACCcLO, F
00A5 0703          M          btfss   STATUS,C
00A6 00F5          M          decf    ACCcHI, F
00A7 0211          M          movf    ACCaHI,W
00A8 00B5          M          subwf   ACCcHI, F
00A9 0503          M          bsf     STATUS,C          ; shift a 1 into b (result)
00AA 0372          M NOGO     rlf     ACCbLO, F
00AB 0373          M          rlf     ACCbHI, F
                   M ;
                 00143        divMac
  0000             M          LOCAL   NOCHK
  0000             M          LOCAL   NOGO
                   M ;
00AC 0403          M          bcf     STATUS,C
00AD 0376          M          rlf     ACCdLO, F
00AE 0377          M          rlf     ACCdHI, F
00AF 0374          M          rlf     ACCcLO, F
00B0 0375          M          rlf     ACCcHI, F
00B1 0211          M          movf    ACCaHI,W
00B2 0095          M          subwf   ACCcHI,W          ; check if a>c
00B3 0743          M          btfss   STATUS,Z
00B4 0AB7          M          goto    NOCHK
00B5 0210          M          movf    ACCaLO,W
00B6 0094          M          subwf   ACCcLO,W          ; if msb equal then check lsb
00B7 0703          M NOCHK    btfss   STATUS,C          ; carry set if c>a
00B8 0AC0          M          goto    NOGO
00B9 0210          M          movf    ACCaLO,W          ; c-a into c
00BA 00B4          M          subwf   ACCcLO, F
00BB 0703          M          btfss   STATUS,C
00BC 00F5          M          decf    ACCcHI, F
00BD 0211          M          movf    ACCaHI,W
00BE 00B5          M          subwf   ACCcHI, F
00BF 0503          M          bsf     STATUS,C          ; shift a 1 into b (result)
00C0 0372          M NOGO     rlf     ACCbLO, F
00C1 0373          M          rlf     ACCbHI, F
                   M ;
                 00144        divMac
  0000             M          LOCAL   NOCHK
  0000             M          LOCAL   NOGO
                   M ;
00C2 0403          M          bcf     STATUS,C
00C3 0376          M          rlf     ACCdLO, F
00C4 0377          M          rlf     ACCdHI, F
00C5 0374          M          rlf     ACCcLO, F
00C6 0375          M          rlf     ACCcHI, F
00C7 0211          M          movf    ACCaHI,W
00C8 0095          M          subwf   ACCcHI,W          ; check if a>c
00C9 0743          M          btfss   STATUS,Z
00CA 0ACD          M          goto    NOCHK
00CB 0210          M          movf    ACCaLO,W
00CC 0094          M          subwf   ACCcLO,W          ; if msb equal then check lsb
00CD 0703          M NOCHK    btfss   STATUS,C          ; carry set if c>a
00CE 0AD6          M          goto    NOGO
00CF 0210          M          movf    ACCaLO,W          ; c-a into c
00D0 00B4          M          subwf   ACCcLO, F
00D1 0703          M          btfss   STATUS,C
00D2 00F5          M          decf    ACCcHI, F
00D3 0211          M          movf    ACCaHI,W
00D4 00B5          M          subwf   ACCcHI, F
00D5 0503          M          bsf     STATUS,C          ; shift a 1 into b (result)
00D6 0372          M NOGO     rlf     ACCbLO, F
00D7 0373          M          rlf     ACCbHI, F
                   M ;
                 00145        divMac
  0000             M          LOCAL   NOCHK
  0000             M          LOCAL   NOGO
```


© 1997 Microchip Technology Inc.

```
                    M ;
00D8 0403           M          bcf     STATUS,C
00D9 0376           M          rlf     ACCdLO, F
00DA 0377           M          rlf     ACCdHI, F
00DB 0374           M          rlf     ACCcLO, F
00DC 0375           M          rlf     ACCcHI, F
00DD 0211           M          movf    ACCaHI,W
00DE 0095           M          subwf   ACCcHI,W          ; check if a>c
00DF 0743           M          btfss   STATUS,Z
00E0 0AE3           M          goto    NOCHK
00E1 0210           M          movf    ACCaLO,W
00E2 0094           M          subwf   ACCcLO,W          ; if msb equal then check lsb
00E3 0703           M NOCHK    btfss   STATUS,C          ; carry set if c>a
00E4 0AEC           M          goto    NOGO
00E5 0210           M          movf    ACCaLO,W          ; c-a into c
00E6 00B4           M          subwf   ACCcLO, F
00E7 0703           M          btfss   STATUS,C
00E8 00F5           M          decf    ACCcHI, F
00E9 0211           M          movf    ACCaHI,W
00EA 00B5           M          subwf   ACCcHI, F
00EB 0503           M          bsf     STATUS,C          ; shift a 1 into b (result)
00EC 0372           M NOGO     rlf     ACCbLO, F
00ED 0373           M          rlf     ACCbHI, F
                    M ;
                  00146        divMac
  0000              M          LOCAL   NOCHK
  0000              M          LOCAL   NOGO
                    M ;
00EE 0403           M          bcf     STATUS,C
00EF 0376           M          rlf     ACCdLO, F
00F0 0377           M          rlf     ACCdHI, F
00F1 0374           M          rlf     ACCcLO, F
00F2 0375           M          rlf     ACCcHI, F
00F3 0211           M          movf    ACCaHI,W
00F4 0095           M          subwf   ACCcHI,W          ; check if a>c
00F5 0743           M          btfss   STATUS,Z
00F6 0AF9           M          goto    NOCHK
00F7 0210           M          movf    ACCaLO,W
00F8 0094           M          subwf   ACCcLO,W          ; if msb equal then check lsb
00F9 0703           M NOCHK    btfss   STATUS,C          ; carry set if c>a
00FA 0B02           M          goto    NOGO
00FB 0210           M          movf    ACCaLO,W          ; c-a into c
00FC 00B4           M          subwf   ACCcLO, F
00FD 0703           M          btfss   STATUS,C
00FE 00F5           M          decf    ACCcHI, F
00FF 0211           M          movf    ACCaHI,W
0100 00B5           M          subwf   ACCcHI, F
0101 0503           M          bsf     STATUS,C          ; shift a 1 into b (result)
0102 0372           M NOGO     rlf     ACCbLO, F
0103 0373           M          rlf     ACCbHI, F
                    M ;
                  00147        divMac
  0000              M          LOCAL   NOCHK
  0000              M          LOCAL   NOGO
                    M ;
0104 0403           M          bcf     STATUS,C
0105 0376           M          rlf     ACCdLO, F
0106 0377           M          rlf     ACCdHI, F
0107 0374           M          rlf     ACCcLO, F
0108 0375           M          rlf     ACCcHI, F
0109 0211           M          movf    ACCaHI,W
010A 0095           M          subwf   ACCcHI,W          ; check if a>c
010B 0743           M          btfss   STATUS,Z
010C 0B0F           M          goto    NOCHK
010D 0210           M          movf    ACCaLO,W
010E 0094           M          subwf   ACCcLO,W          ; if msb equal then check lsb
```


© 1997 Microchip Technology Inc.

```
010F 0703          M NOCHK   btfss   STATUS,C        ; carry set if c>a
0110 0B18          M         goto    NOGO
0111 0210          M         movf    ACCaLO,W        ; c-a into c
0112 00B4          M         subwf   ACCcLO, F
0113 0703          M         btfss   STATUS,C
0114 00F5          M         decf    ACCcHI, F
0115 0211          M         movf    ACCaHI,W
0116 00B5          M         subwf   ACCcHI, F
0117 0503          M         bsf     STATUS,C        ; shift a 1 into b (result)
0118 0372          M NOGO    rlf     ACCbLO, F
0119 0373          M         rlf     ACCbHI, F
                   M ;
                 00148       divMac
  0000             M         LOCAL   NOCHK
  0000             M         LOCAL   NOGO
                   M ;
011A 0403          M         bcf     STATUS,C
011B 0376          M         rlf     ACCdLO, F
011C 0377          M         rlf     ACCdHI, F
011D 0374          M         rlf     ACCcLO, F
011E 0375          M         rlf     ACCcHI, F
011F 0211          M         movf    ACCaHI,W
0120 0095          M         subwf   ACCcHI,W         ; check if a>c
0121 0743          M         btfss   STATUS,Z
0122 0B25          M         goto    NOCHK
0123 0210          M         movf    ACCaLO,W
0124 0094          M         subwf   ACCcLO,W        ; if msb equal then check lsb
0125 0703          M NOCHK   btfss   STATUS,C        ; carry set if c>a
0126 0B2E          M         goto    NOGO
0127 0210          M         movf    ACCaLO,W        ; c-a into c
0128 00B4          M         subwf   ACCcLO, F
0129 0703          M         btfss   STATUS,C
012A 00F5          M         decf    ACCcHI, F
012B 0211          M         movf    ACCaHI,W
012C 00B5          M         subwf   ACCcHI, F
012D 0503          M         bsf     STATUS,C        ; shift a 1 into b (result)
012E 0372          M NOGO    rlf     ACCbLO, F
012F 0373          M         rlf     ACCbHI, F
                   M ;
                 00149       divMac
  0000             M         LOCAL   NOCHK
  0000             M         LOCAL   NOGO
                   M ;
0130 0403          M         bcf     STATUS,C
0131 0376          M         rlf     ACCdLO, F
0132 0377          M         rlf     ACCdHI, F
0133 0374          M         rlf     ACCcLO, F
0134 0375          M         rlf     ACCcHI, F
0135 0211          M         movf    ACCaHI,W
0136 0095          M         subwf   ACCcHI,W        ; check if a>c
0137 0743          M         btfss   STATUS,Z
0138 0B3B          M         goto    NOCHK
0139 0210          M         movf    ACCaLO,W
013A 0094          M         subwf   ACCcLO,W        ; if msb equal then check lsb
013B 0703          M NOCHK   btfss   STATUS,C        ; carry set if c>a
013C 0B44          M         goto    NOGO
013D 0210          M         movf    ACCaLO,W        ; c-a into c
013E 00B4          M         subwf   ACCcLO, F
013F 0703          M         btfss   STATUS,C
0140 00F5          M         decf    ACCcHI, F
0141 0211          M         movf    ACCaHI,W
0142 00B5          M         subwf   ACCcHI, F
0143 0503          M         bsf     STATUS,C        ; shift a 1 into b (result)
0144 0372          M NOGO    rlf     ACCbLO, F
0145 0373          M         rlf     ACCbHI, F
                   M ;
```


© 1997 Microchip Technology Inc.

```
                00150          divMac
  0000              M          LOCAL   NOCHK
  0000              M          LOCAL   NOGO
                    M ;
0146 0403           M          bcf     STATUS,C
0147 0376           M          rlf     ACCdLO, F
0148 0377           M          rlf     ACCdHI, F
0149 0374           M          rlf     ACCcLO, F
014A 0375           M          rlf     ACCcHI, F
014B 0211           M          movf    ACCaHI,W
014C 0095           M          subwf   ACCcHI,W          ; check if a>c
014D 0743           M          btfss   STATUS,Z
014E 0B51           M          goto    NOCHK
014F 0210           M          movf    ACCaLO,W
0150 0094           M          subwf   ACCcLO,W          ; if msb equal then check lsb
0151 0703           M NOCHK    btfss   STATUS,C          ; carry set if c>a
0152 0B5A           M          goto    NOGO
0153 0210           M          movf    ACCaLO,W          ; c-a into c
0154 00B4           M          subwf   ACCcLO, F
0155 0703           M          btfss   STATUS,C
0156 00F5           M          decf    ACCcHI, F
0157 0211           M          movf    ACCaHI,W
0158 00B5           M          subwf   ACCcHI, F
0159 0503           M          bsf     STATUS,C          ; shift a 1 into b (result)
015A 0372           M NOGO     rlf     ACCbLO, F
015B 0373           M          rlf     ACCbHI, F
                    M ;
                00151          divMac
  0000              M          LOCAL   NOCHK
  0000              M          LOCAL   NOGO
                    M ;
015C 0403           M          bcf     STATUS,C
015D 0376           M          rlf     ACCdLO, F
015E 0377           M          rlf     ACCdHI, F
015F 0374           M          rlf     ACCcLO, F
0160 0375           M          rlf     ACCcHI, F
0161 0211           M          movf    ACCaHI,W
0162 0095           M          subwf   ACCcHI,W          ; check if a>c
0163 0743           M          btfss   STATUS,Z
0164 0B67           M          goto    NOCHK
0165 0210           M          movf    ACCaLO,W
0166 0094           M          subwf   ACCcLO,W          ; if msb equal then check lsb
0167 0703           M NOCHK    btfss   STATUS,C          ; carry set if c>a
0168 0B70           M          goto    NOGO
0169 0210           M          movf    ACCaLO,W          ; c-a into c
016A 00B4           M          subwf   ACCcLO, F
016B 0703           M          btfss   STATUS,C
016C 00F5           M          decf    ACCcHI, F
016D 0211           M          movf    ACCaHI,W
016E 00B5           M          subwf   ACCcHI, F
016F 0503           M          bsf     STATUS,C          ; shift a 1 into b (result)
0170 0372           M NOGO     rlf     ACCbLO, F
0171 0373           M          rlf     ACCbHI, F
                    M ;
                00152 ;
                00153      IF    SIGNED
                00154          btfss   sign,MSB          ; check sign if negative
                00155          retlw   0
                00156          goto    neg_B             ; negate ACCa (-ACCa -> ACCa)
                00157      ELSE
0172 0800       00158          retlw   0
                00159      ENDIF
                00160 ;
                00161 ;*****************************************************************
                00162 ;  Assemble this section only if Signed Arithmetic Needed
                00163 ;
```


© 1997 Microchip Technology Inc.

```
                00164      IF    SIGNED
                00165 ;
                00166 S_SIGN  movf    ACCaHI,W
                00167         xorwf   ACCbHI,W
                00168         movwf   sign
                00169         btfss   ACCbHI,MSB        ; if MSB set go & negate ACCb
                00170         goto    chek_A
                00171 ;
                00172         comf    ACCbLO            ; negate ACCb
                00173         incf    ACCbLO
                00174         btfsc   STATUS,Z
                00175         decf    ACCbHI
                00176         comf    ACCbHI
                00177 ;
                00178 chek_A  btfss   ACCaHI,MSB        ; if MSB set go & negate ACCa
                00179         retlw   0
                00180         goto    neg_A
                00181 ;
                00182      ENDIF
                00183 ;
                00184 ;*****************************************************************
                00185 ;                          Test Program
                00186 ;*****************************************************************
                00187 ;    Load constant values to ACCa & ACCb for testing
                00188 ;
0173 0C01       00189 main    movlw   1
0174 0031       00190         movwf   ACCaHI
0175 0CFF       00191         movlw   0FF             ; loads ACCa = 01FF
0176 0030       00192         movwf   ACCaLO
                00193 ;
0177 0C7F       00194         movlw   07F
0178 0033       00195         movwf   ACCbHI
0179 0CFF       00196         movlw   0FF             ; loads ACCb = 7FFF
017A 0032       00197         movwf   ACCbLO
                00198
017B 090F       00199         call    D_divF          ; remainder in ACCc. Here ACCb = 0040 &
                00200                                                   ; ACCc=003F
017C 0B7C       00201 self    goto    self
                00202 ;
01FF            00203         org     PIC54
01FF 0B73       00204         goto    main
                00205         END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXX----
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   382
Program Memory Words Free:   130


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX L:DOUBLE PRECISION DIVISION LISTING (FAST)

```
MPASM 01.40 Released          DBL_DIVS.ASM   1-16-1997  12:51:51          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 16C54, n = 66
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;                       Double Precision Division
                     00005 ;
                     00006 ;               ( Optimized for Code Size : Looped Code )
                     00007 ;
                     00008 ;*****************************************************************;
                     00009 ; Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
                     00010 ;                                            Remainder in ACCc (16 bits)
                     00011 ;     (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
                     00012 ;     (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
                     00013 ;     (c) CALL D_div
                     00014 ;     (d) The 16 bit result is in location ACCbHI & ACCbLO
                     00015 ;     (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
                     00016 ;
                     00017 ;   Performance :
                     00018 ;               Program Memory  :       037
                     00019 ;               Clock Cycles    :       310
                     00020 ;
                     00021 ;        NOTE :
                     00022 ;               The performance specs are for Unsigned arithmetic
                     00023 ;               ( i.e,with "SIGNED equ  FALSE ").
                     00024 ;
                     00025 ;
                     00026 ;       Program:          DBL_DIVS.ASM
                     00027 ;       Revision Date:
                     00028 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00029 ;
                     00030 ;*****************************************************************;
                     00031 ;
  00000010           00032 ACCaLO  equ     10
  00000011           00033 ACCaHI  equ     11
  00000012           00034 ACCbLO  equ     12
  00000013           00035 ACCbHI  equ     13
  00000014           00036 ACCcLO  equ     14
  00000015           00037 ACCcHI  equ     15
  00000016           00038 ACCdLO  equ     16
  00000017           00039 ACCdHI  equ     17
  00000018           00040 temp    equ     18
  00000019           00041 sign    equ     19
                     00042 ;
                     00043         include "p16c5x.inc"
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology, Inc.
                     00224         LIST
                     00044
  000001FF           00045 PIC54   equ     1FFH    ; Define Reset Vector
  00000001           00046 TRUE    equ     1
  00000000           00047 FALSE   equ     0
                     00048
0000                 00049         org     0
                     00050 ;*****************************************************************
  00000000           00051 SIGNED  equ     FALSE           ; Set This To 'TRUE' if the routines
```


© 1997 Microchip Technology Inc.

```
                    00052 ;                             ; for Multiplication & Division needs
                    00053 ;                             ; to be assembled as Signed Integer
                    00054 ;                             ; Routines. If 'FALSE' the above two
                    00055 ;                             ; routines ( D_mpy & D_div ) use
                    00056 ;                             ; unsigned arithmetic.
                    00057 ;*****************************************************************
                    00058 ;       Double Precision Divide ( 16/16 -> 16 )
                    00059 ;
                    00060 ; (ACCb/ACCa -> ACCb with remainder in ACCc) : 16 bit output
                    00061 ; with Quotiont in ACCb (ACCbHI,ACCbLO) and Remainder in ACCc
                    00062 ; (ACCcHI,ACCcLO).
                    00063 ; NOTE: Before calling this routine, the user should make sure that
                    00064 ;       the Numerator(ACCb) is greater than Denominator(ACCa). If
                    00065 ;       the case is not true, the user should scale either Numerator
                    00066 ;       or Denominator or both such that Numerator is greater than
                    00067 ;       the Denominator.
                    00068 ;
                    00069 ;
0000                00070 D_divS
                    00071 ;
                    00072       IF   SIGNED
                    00073       CALL    S_SIGN
                    00074       ENDIF
                    00075 ;
0000 091C           00076           call    setup
0001 0075           00077           clrf    ACCcHI
0002 0074           00078           clrf    ACCcLO
0003 0403           00079 dloop     bcf     STATUS,C
0004 0376           00080           rlf     ACCdLO, F
0005 0377           00081           rlf     ACCdHI, F
0006 0374           00082           rlf     ACCcLO, F
0007 0375           00083           rlf     ACCcHI, F
0008 0211           00084           movf    ACCaHI,W
0009 0095           00085           subwf   ACCcHI,W        ; check if a>c
000A 0743           00086           btfss   STATUS,Z
000B 0A0E           00087           goto    nochk
000C 0210           00088           movf    ACCaLO,W
000D 0094           00089           subwf   ACCcLO,W        ; if msb equal then check lsb
000E 0703           00090 nochk     btfss   STATUS,C        ; carry set if c>a
000F 0A17           00091           goto    nogo
0010 0210           00092           movf    ACCaLO,W        ; c-a into c
0011 00B4           00093           subwf   ACCcLO, F
0012 0703           00094           btfss   STATUS,C
0013 00F5           00095           decf    ACCcHI, F
0014 0211           00096           movf    ACCaHI,W
0015 00B5           00097           subwf   ACCcHI, F
0016 0503           00098           bsf     STATUS,C        ; shift a 1 into b (result)
0017 0372           00099 nogo      rlf     ACCbLO, F
0018 0373           00100           rlf     ACCbHI, F
0019 02F8           00101           decfsz  temp, F         ; loop untill all bits checked
001A 0A03           00102           goto    dloop
                    00103 ;
                    00104       IF    SIGNED
                    00105           btfss   sign,MSB        ; check sign if negative
                    00106           retlw   0
                    00107           goto    neg_B           ; negate ACCa ( -ACCa -> ACCa )
                    00108       ELSE
001B 0800           00109           retlw   0
                    00110       ENDIF
                    00111 ;
                    00112 ;*****************************************************************
                    00113 ;
001C 0C10           00114 setup     movlw   .16             ; for 16 shifts
001D 0038           00115           movwf   temp
001E 0213           00116           movf    ACCbHI,W        ; move ACCb to ACCd
001F 0037           00117           movwf   ACCdHI
```

```
0020 0212           00118         movf    ACCbLO,W
0021 0036           00119         movwf   ACCdLO
0022 0073           00120         clrf    ACCbHI
0023 0072           00121         clrf    ACCbLO
0024 0800           00122         retlw   0
                    00123 ;
                    00124 ;*****************************************************************
                    00125 ;
0025 0270           00126 neg_A   comf    ACCaLO, F       ; negate ACCa ( -ACCa -> ACCa )
0026 02B0           00127         incf    ACCaLO, F
0027 0643           00128         btfsc   STATUS,Z
0028 00F1           00129         decf    ACCaHI, F
0029 0271           00130         comf    ACCaHI, F
002A 0800           00131         retlw   0
                    00132 ;
                    00133 ;*****************************************************************
                    00134 ;  Assemble this section only if Signed Arithmetic Needed
                    00135 ;
                    00136       IF    SIGNED
                    00137 ;
                    00138 S_SIGN  movf    ACCaHI,W
                    00139         xorwf   ACCbHI,W
                    00140         movwf   sign
                    00141         btfss   ACCbHI,MSB        ; if MSB set go & negate ACCb
                    00142         goto    chek_A
                    00143 ;
                    00144         comf    ACCbLO            ; negate ACCb
                    00145         incf    ACCbLO
                    00146         btfsc   STATUS,Z
                    00147         decf    ACCbHI
                    00148         comf    ACCbHI
                    00149 ;
                    00150 chek_A  btfss   ACCaHI,MSB        ; if MSB set go & negate ACCa
                    00151         retlw   0
                    00152         goto    neg_A
                    00153 ;
                    00154       ENDIF
                    00155 ;
                    00156 ;*****************************************************************
                    00157 ;                         Test Program
                    00158 ;*****************************************************************
                    00159 ;    Load constant values to ACCa & ACCb for testing
                    00160 ;
002B 0C01           00161 main    movlw   1
002C 0031           00162         movwf   ACCaHI
002D 0CFF           00163         movlw   0FF             ; loads ACCa = 01FF
002E 0030           00164         movwf   ACCaLO
                    00165 ;
002F 0C7F           00166         movlw   07F
0030 0033           00167         movwf   ACCbHI
0031 0CFF           00168         movlw   0FF             ; loads ACCb = 7FFF
0032 0032           00169         movwf   ACCbLO
                    00170 ;
0033 0900           00171         call    D_divS          ; remainder in ACCc. Here ACCb =0040 &
ACCc=003F
                    00172 ;
0034 0A34           00173 self    goto    self
                    00174 ;
01FF                00175         org     PIC54
01FF 0A2B           00176         goto    main
                    00177         END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)


0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXX-----------
01C0 : ---------------- ---------------- ---------------- ---------------X
```


© 1997 Microchip Technology Inc.

```
All other memory blocks unused.

Program Memory Words Used:     54
Program Memory Words Free:    458


Errors   :      0
Warnings :      0 reported,      0 suppressed
Messages :      0 reported,      0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX M:

```
    LIST
; P16C5X.INC  Standard Header File, Version 3.30    Microchip Technology, Inc.
    NOLIST

; This header file defines configurations, registers, and other useful bits of
; information for the 16C5X microcontrollers.  These names are taken to match
; the data sheets as closely as possible.  The microcontrollers included
; in this file are:

;    16C52
;    16C54
;    16CR54
;    16C54A
;    16CR54A
;    16C55
;    16C56
;    16C57
;    16CR57A
;    16CR57B
;    16C58A
;    16CR58A

; There is one group of symbols that is valid for all microcontrollers.
; Each microcontroller in this family also has its own section of special
; symbols.  Note that the processor must be selected before this file is
; included.  The processor may be selected the following ways:

;       1. Command line switch:
;               C:\ MPASM MYFILE.ASM /P16C54A
;       2. LIST directive in the source file
;               LIST   P=16C54A
;       3. Processor Type entry in the MPASM full-screen interface

;==========================================================================
;
;      Revision History
;
;==========================================================================

;Rev:    Date:    Reason:

;3.30   07/16/96 Aligned processors with MPASM v1.40
;3.2004/09/96 Added 16C54B, 16CR56B, 16C58B
;3.10   12/14/95 Added 16C52
;3.01   11/29/95 Removed 16CR55
;3.00   10/16/95 Added new processors for MPASM v1.30
;2.04   07/26/95 Reformatted for readability
;2.03   06/21/95 Removed leading spaces

;==========================================================================
;
;       Generic Definitions
;
;==========================================================================

W                               EQU     H'0000'
F                               EQU     H'0001'

;----- Register Files ------------------------------------------------------
```


© 1997 Microchip Technology Inc.

```
INDF                            EQU     H'0000'
TMR0                            EQU     H'0001'
PCL                             EQU     H'0002'
STATUS                          EQU     H'0003'
FSR                             EQU     H'0004'
PORTA                           EQU     H'0005'
PORTB                           EQU     H'0006'

;----- STATUS Bits --------------------------------------------------

PA2                             EQU     H'0007'
PA1                             EQU     H'0006'
PA0                             EQU     H'0005'
NOT_TO                          EQU     H'0004'
NOT_PD                          EQU     H'0003'
Z                               EQU     H'0002'
DC                              EQU     H'0001'
C                               EQU     H'0000'

;----- OPTION Bits --------------------------------------------------

T0CS                            EQU     H'0005'
T0SE                            EQU     H'0004'
PSA                             EQU     H'0003'
PS2                             EQU     H'0002'
PS1                             EQU     H'0001'
PS0                             EQU     H'0000'

;==========================================================================
;
;       Processor-dependent Definitions
;
;==========================================================================

        IFDEF __16C52
           __MAXRAM H'01F'
           #define __CONFIG_2
        ENDIF

;--------------------------------------------------------------------------

        IFDEF __16C54
           __MAXRAM H'01F'
           #define __CONFIG_0
        ENDIF

;--------------------------------------------------------------------------

        IFDEF __16CR54
           __MAXRAM H'01F'
           #define __CONFIG_0
        ENDIF

;--------------------------------------------------------------------------

        IFDEF __16C54A
           __MAXRAM H'01F'
           #define __CONFIG_0
        ENDIF

;--------------------------------------------------------------------------

        IFDEF __16CR54A
           __MAXRAM H'01F'
           #define __CONFIG_1
        ENDIF
```


© 1997 Microchip Technology Inc.

```
;-------------------------------------------------------------------------

        IFDEF __16C55
                                                    ; Register Files
PORTC                       EQU     H'0007'
           __MAXRAM H'01F'
           #define __CONFIG_0
        ENDIF

;-------------------------------------------------------------------------

        IFDEF __16C56
           __MAXRAM H'01F'
           #define __CONFIG_0
        ENDIF

;-------------------------------------------------------------------------

        IFDEF __16C57
                                                    ; Register Files
PORTC                       EQU     H'0007'
           __MAXRAM H'07F'
           #define __CONFIG_0
        ENDIF

;-------------------------------------------------------------------------

        IFDEF __16CR57A
                                                    ; Register Files
PORTC                       EQU     H'0007'
           __MAXRAM H'07F'
           #define __CONFIG_0
        ENDIF

;-------------------------------------------------------------------------

        IFDEF __16CR57B
                                                    ; Register Files
PORTC                       EQU     H'0007'
           __MAXRAM H'07F'
           #define __CONFIG_1
        ENDIF

;-------------------------------------------------------------------------

        IFDEF __16C58A
           __MAXRAM H'07F'
           #define __CONFIG_0
        ENDIF

;-------------------------------------------------------------------------

        IFDEF __16CR58A
           __MAXRAM H'07F'
           #define __CONFIG_1
        ENDIF

;=========================================================================
;
;       Configuration Bits
;
;=========================================================================

        IFDEF __CONFIG_0
_CP_ON                      EQU     H'0FF7'
```


© 1997 Microchip Technology Inc.

```
_CP_OFF                         EQU     H'0FFF'
_WDT_ON                         EQU     H'0FFF'
_WDT_OFF                        EQU     H'0FFB'
_LP_OSC                         EQU     H'0FFC'
_XT_OSC                         EQU     H'0FFD'
_HS_OSC                         EQU     H'0FFE'
_RC_OSC                         EQU     H'0FFF'
          #undefine __CONFIG_0
        ENDIF


        IFDEF __CONFIG_1
_CP_ON                          EQU     H'0007'
_CP_OFF                         EQU     H'0FFF'
_WDT_ON                         EQU     H'0FFF'
_WDT_OFF                        EQU     H'0FFB'
_LP_OSC                         EQU     H'0FFC'
_XT_OSC                         EQU     H'0FFD'
_HS_OSC                         EQU     H'0FFE'
_RC_OSC                         EQU     H'0FFF'
          #undefine __CONFIG_1
        ENDIF

        IFDEF __CONFIG_2
_CP_ON                          EQU     H'0FF7'
_CP_OFF                         EQU     H'0FFF'
_XT_OSC                         EQU     H'0FFD'
_RC_OSC                         EQU     H'0FFF'
          #undefine __CONFIG_2
        ENDIF

    LIST
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX N:INCLUDE FILE FOR FIXED POINT ROUTINE

```
                processor16C71

;       define assembler constants

B0              equ 0
B1              equ 1
B2              equ 2
B3              equ 3
B4              equ 4
B5              equ 5

MSB             equ 7
LSB             equ 0

W               equ 0

;       define special function registers

                cblock 0x00; page 0 registers
                        INDF,RTCC,PCL,STATUS,FSR,TRISA,TRISB,ZZZZ,
                        ADCON0,ADRES,PCLATH,INTCON
                endc

                cblock 0x00; page 1 registers
                        INDF,OPTION,PCL,STATUS,FSR,PORTA,PORTB,ZZZZ,
                        ADCON1,ADRES,PCLATH,INTCON
                endc

;       define beginning of general purpose RAM

RAMSTART        equ     0x0C
RAMSTOP         equ     0x2F

;       define commonly used bits

;       STATUS bit definitions

                #define_C   STATUS,0
                #define_DC  STATUS,1
                #define_Z   STATUS,2
                #define_PD  STATUS,3
                #define_TO  STATUS,4
                #define_RP0 STATUS,5
                #define_PA0 STATUS,5
                #define_RP1 STATUS,6
                #define_PA1 STATUS,6
                #define_IRP STATUS,7
                #define_PA2 STATUS,7
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX O:

```
;       16/8 PIC16 FIXED POINT DIVIDE ROUTINES  VERSION 1.5

;       Input:  fixed point arguments in AARG and BARG

;       Output: quotient AARG/BARG followed by remainder in REM

;       All timings are worst case cycle counts

;       It is useful to note that the additional routine FXD1507U
;       can be called in a signed divide application in the special case
;       where AARG > 0 and BARG > 0, thereby offering some improvement in
;       performance.

;         Routine             Clocks     Function

;       FXD1608S    188 16 bit/8 bit -> 16.08 signed fixed point divide

;       FXD1608U    294 16 bit/8 bit -> 16.08 unsigned fixed point divide

;       FXD1607U    174 16 bit/7 bit -> 16.07 unsigned fixed point divide

;       FXD1507U    166 15 bit/7 bit -> 15.07 unsigned fixed point divide

;       The above timings are based on the looped macros. If space permits,
;       approximately 41-50 clocks can be saved by using the unrolled macros.


                list    r=dec,x=on,t=off,p=16C71

                include <PIC16.INC>

;*****************************************************************************************
;*****************************************************************************************

;       Define divide register variables

ACC             equ     0x0D    ; most significant byte of contiguous 4 byte accumulator

SIGN            equ     0x13    ; save location for sign in MSB

TEMP            equ     0x19    ; temporary storage

;       Define binary operation arguments

AARG            equ     0x0D    ; most significant byte of argument A

BARG            equ     0x16    ; most significant byte of argument B

REM             equ     0x11    ; most significant byte of remainder

LOOPCOUNT       equ     0x14    ; loop counter

;       Note:   ( AARG+B0, AARG+B1 )  and  ( ACC+B0, ACC+B1)
;               reference the same storage locations, and similarly for
;               ( REM+B0, REM+B1 ) and ( ACC+B4, ACC+B5 )


;*****************************************************************************************
;*****************************************************************************************

;       16/08 BIT Division Macros
```


© 1997 Microchip Technology Inc.

```
SDIV1608L       macro

;       Max Timing:     3+5+2+5*11+10+10+6*11+10+2 = 163 clks

;       Min Timing:     3+5+2+5*11+10+10+6*11+10+2 = 163 clks

;       PM: 42                                  DM: 5

                MOVF            BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                ADDWF           REM+B0
                RLF             ACC+B0

                MOVLW           6
                MOVWF           LOOPCOUNT

LOOPS1608A      RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS1608A

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPS1608B      RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS1608B

                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0

                endm

UDIV1608L  macro
```


© 1997 Microchip Technology Inc.

```
;       Max Timing: 2+7*12+11+3+7*24+23 = 291 clks

;       Min Timing: 2+7*11+10+3+7*17+16 = 227 clks

;       PM: 39                                  DM: 7

                MOVLW           8
                MOVWF           LOOPCOUNT

LOOPU1608A      RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                SUBWF           REM+B0

                BTFSC           _C
                GOTO            UOK68A
                ADDWF           REM+B0
                BCF             _C
UOK68A          RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1608A

                CLRF            TEMP

                MOVLW           8
                MOVWF           LOOPCOUNT

LOOPU1608B      RLF             ACC+B1,W
                RLF             REM+B0
                RLF             TEMP
                MOVF            BARG+B0,W
                SUBWF           REM+B0
                CLRF            ACC+B5
                CLRW
                BTFSS           _C
                INCFSZ          ACC+B5,W
                SUBWF           TEMP

                BTFSC           _C
                GOTO            UOK68B
                MOVF            BARG+B0,W
                ADDWF           REM+B0
                CLRF            ACC+B5
                CLRW
                BTFSC           _C
                INCFSZ          ACC+B5,W
                ADDWF           TEMP

                BCF             _C
UOK68B          RLF             ACC+B1

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1608B

                endm

UDIV1607L       macro

;       Max Timing:     7+6*11+10+10+6*11+10+2 = 171 clks

;       Min Timing:     7+6*11+10+10+6*11+10+2 = 171 clks

;       PM: 39                                  DM: 5

                RLF             ACC+B0,W
                RLF             REM+B0
```


© 1997 Microchip Technology Inc.

```
                MOVF            BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU1607A      RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1607A

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU1607B      RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1607B

                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0

                endm

UDIV1507L       macro

;       Max Timing:     3+5+2+5*11+10+10+6*11+10+2 = 163 clks

;       Min Timing:     3+5+2+5*11+10+10+6*11+10+2 = 163 clks

;       PM: 42                                  DM: 5

                MOVF            BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                ADDWF           REM+B0
                RLF             ACC+B0
```


© 1997 Microchip Technology Inc.

```
                MOVLW           6
                MOVWF           LOOPCOUNT

LOOPU1507A      RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1507A

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU1507B      RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1507B

                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0

                endm

SDIV1608        macro

;       Max Timing:     3+5+14*8+2 = 122 clks

;       Min Timing:     3+5+14*8+2 = 122 clks

;       PM: 122                                 DM: 4

                variable i

                MOVF            BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                ADDWF           REM+B0
                RLF             ACC+B0
```


© 1997 Microchip Technology Inc.

```
                i = 2

                while i < 8

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B0

                i=i+1

                endw

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                i = 9

                while i < 16

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                i=i+1

                endw

                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0

                endm

UDIV1608  macro

;       restore = 9/21 clks,  nonrestore = 8/14 clks

;       Max Timing: 8*9+1+8*21 = 241 clks

;       Min Timing: 8*8+1+8*14 = 177 clks

;       PM: 241                                 DM: 6

                variable        i

                i = 0

                while i < 8

                RLF             ACC+B0,W
```


© 1997 Microchip Technology Inc.

```
                RLF             REM+B0
                MOVF            BARG+B0,W
                SUBWF           REM+B0

                BTFSC           _C
                GOTO            UOK68#v(i)
                ADDWF           REM+B0
                BCF             _C
UOK68#v(i)      RLF             ACC+B0

                i=i+1

                endw

                CLRF            TEMP

                i = 8

                while i < 16

                RLF             ACC+B1,W
                RLF             REM+B0
                RLF             TEMP
                MOVF            BARG+B0,W
                SUBWF           REM+B0
                CLRF            ACC+B5
                CLRW
                BTFSS           _C
                INCFSZ          ACC+B5,W
                SUBWF           TEMP

                BTFSC           _C
                GOTO            UOK68#v(i)
                MOVF            BARG+B0,W
                ADDWF           REM+B0
                CLRF            ACC+B5
                CLRW
                BTFSC           _C
                INCFSZ          ACC+B5,W
                ADDWF           TEMP

                BCF             _C
UOK68#v(i)      RLF             ACC+B1

                i=i+1

                endw

                endm

UDIV1607        macro

;       Max Timing:     5+15*8+2 = 127 clks

;       Min Timing:     5+15*8+2 = 127 clks

;       PM: 127                                 DM: 4

                variable i

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                i = 1
```


© 1997 Microchip Technology Inc.

```
                while i < 8

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B0

                i=i+1

                endw

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B0,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B0,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                i = 9

                while i < 16

                RLF             ACC+B1,W
                RLF             REM+B0
                MOVF            BARG+B0,W

                BTFSC           ACC+B1,LSB
                SUBWF           REM+B0
                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0
                RLF             ACC+B1

                i=i+1

                endw

                BTFSS           ACC+B1,LSB
                ADDWF           REM+B0

                endm

UDIV1507        macro

;       Max Timing:     3+5+14*8+2 = 122 clks

;       Min Timing:     3+5+14*8+2 = 122 clks

;       PM: 122                                 DM: 4

                variable i

                MOVF            BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                RLF             ACC+B0,W
                RLF             REM+B0
                MOVF            BARG+B0,W
                ADDWF           REM+B0
                RLF             ACC+B0
```


© 1997 Microchip Technology Inc.

```
        i = 2

        while i < 8

        RLF             ACC+B0,W
        RLF             REM+B0
        MOVF            BARG+B0,W

        BTFSC           ACC+B0,LSB
        SUBWF           REM+B0
        BTFSS           ACC+B0,LSB
        ADDWF           REM+B0
        RLF             ACC+B0

        i=i+1

        endw

        RLF             ACC+B1,W
        RLF             REM+B0
        MOVF            BARG+B0,W

        BTFSC           ACC+B0,LSB
        SUBWF           REM+B0
        BTFSS           ACC+B0,LSB
        ADDWF           REM+B0
        RLF             ACC+B1

        i = 9

        while i < 16

        RLF             ACC+B1,W
        RLF             REM+B0
        MOVF            BARG+B0,W

        BTFSC           ACC+B1,LSB
        SUBWF           REM+B0
        BTFSS           ACC+B1,LSB
        ADDWF           REM+B0
        RLF             ACC+B1

        i=i+1

        endw

        BTFSS           ACC+B1,LSB
        ADDWF           REM+B0

        endm
;*****************************************************************************************
;*****************************************************************************************

;       16/8 Bit Signed Fixed Point Divide 16/8 -> 16.08

;       Input:  16 bit signed fixed point dividend in AARG+B0, AARG+B1
;               8 bit signed fixed point divisor in BARG+B0

;       Use:    CALL    FXD1608S

;       Output: 16 bit signed fixed point quotient in AARG+B0, AARG+B1
;               8 bit signed fixed point remainder in REM+B0

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     10+163+3 = 176 clks             A > 0, B > 0
```


© 1997 Microchip Technology Inc.

```
;                                       11+163+11 = 185 clks            A > 0, B < 0
;                                       14+163+11 = 188 clks            A < 0, B > 0
;                                       15+163+3 = 181 clks             A < 0, B < 0

;       Min Timing:     10+163+3 = 176 clks             A > 0, B > 0
;                                       11+163+11 = 185 clks            A > 0, B < 0
;                                       14+163+11 = 188 clks            A < 0, B > 0
;                                       15+163+3 = 181 clks             A < 0, B < 0

;       PM: 15+42+10 = 67               DM: 6

FXD1608S        MOVF            AARG+B0,W
                XORWF           BARG+B0,W
                MOVWF           SIGN

                BTFSS           BARG+B0,MSB         ; if MSB set go & negate BARG
                GOTO            CA1608S

                COMF            BARG+B0
                INCF            BARG+B0

CA1608S         BTFSS           AARG+B0,MSB         ; if MSB set go & negate ACCa
                GOTO            C1608S

                COMF            AARG+B1
                INCF            AARG+B1
                BTFSC           _Z
                DECF            AARG+B0
                COMF            AARG+B0

C1608S          CLRF            REM+B0

                SDIV1608L

                BTFSS           SIGN,MSB            ; negate (ACCc,ACCd)
                RETLW           0x00

                COMF            AARG+B1
                INCF            AARG+B1
                BTFSC           _Z
                DECF            AARG+B0
                COMF            AARG+B0

                COMF            REM+B0
                INCF            REM+B0

                RETLW           0x00


;*****************************************************************************************
;*****************************************************************************************

;       16/8 Bit Unsigned Fixed Point Divide 16/8 -> 16.08

;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               16 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD1608U

;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               16 bit unsigned fixed point remainder in REM+B0

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     1+291+2 = 294 clks

;       Min Timing:     1+227+2 = 230 clks
```


© 1997 Microchip Technology Inc.

```
;       PM: 1+39+1 = 41          DM: 7

FXD1608U        CLRF            REM+B0

                UDIV1608L

                RETLW           0x00

;*****************************************************************************************
;*****************************************************************************************

;       16/7 Bit Unsigned Fixed Point Divide 16/7 -> 16.07

;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               7 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD1607U

;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               7 bit unsigned fixed point remainder in REM+B0

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     1+171+2 = 174 clks

;       Min Timing:     1+171+2 = 174 clks

;       PM: 1+39+1 = 41          DM: 5

FXD1607U        CLRF            REM+B0

                UDIV1607L

                RETLW           0x00

;*****************************************************************************************
;*****************************************************************************************

;       15/7 Bit Unsigned Fixed Point Divide 15/7 -> 15.07

;       Input:  15 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               7 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD1507U

;       Output: 15 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               7 bit unsigned fixed point remainder in REM+B0

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     1+163+2 = 166 clks

;       Min Timing:     1+163+2 = 166 clks

;       PM: 1+42+1 = 44          DM: 5

FXD1507U        CLRF            REM+B0

                UDIV1507L

                RETLW           0x00

                END
;*****************************************************************************************
;*****************************************************************************************
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX P:16/16 FIXED POINT DIVIDE ROUTINES

```
;       16/16 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.5

;       Input:  fixed point arguments in AARG and BARG

;       Output: quotient AARG/BARG followed by remainder in REM

;       All timings are worst case cycle counts

;       It is useful to note that the additional routine FXD1515U
;       can be called in a signed divide application in the special case
;       where AARG > 0 and BARG > 0, thereby offering some improvement in
;       performance.

;         Routine             Clocks     Function

;       FXD1616S    319 16 bit/16 bit -> 16.16 signed fixed point divide

;       FXD1616U    373 16 bit/16 bit -> 16.16 unsigned fixed point divide

;       FXD1515U    294 15 bit/15 bit -> 15.15 unsigned fixed point divide

;       The above timings are based on the looped macros. If space permits,
;       approximately 65-69 clocks can be saved by using the unrolled macros.


                list    r=dec,x=on,t=off,p=16C71

                include <PIC16.INC>

;*****************************************************************************************
;*****************************************************************************************

;       Define divide register variables

ACC             equ     0x0D    ; most significant byte of contiguous 4 byte accumulator

SIGN            equ     0x13    ; save location for sign in MSB

TEMP            equ     0x19    ; temporary storage

;       Define binary operation arguments

AARG            equ     0x0D    ; most significant byte of argument A

BARG            equ     0x16    ; most significant byte of argument B

REM             equ     0x11    ; most significant byte of remainder

LOOPCOUNT       equ     0x14    ; loop counter

;       Note:   ( AARG+B0, AARG+B1 )  and  ( ACC+B0, ACC+B1)
;               reference the same storage locations, and similarly for
;               ( REM+B0, REM+B1 ) and ( ACC+B4, ACC+B5 )


;*****************************************************************************************
;*****************************************************************************************

;       16/16 Bit Division Macros
```


© 1997 Microchip Technology Inc.

```
SDIV1616L       macro

;       Max Timing:     13+14*18+17+8 = 290 clks

;       Min Timing:     13+14*16+15+3 = 255 clks

;       PM: 42                                  DM: 7

                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B1
                RLF             ACC+B0

                MOVLW           15
                MOVWF           LOOPCOUNT

LOOPS1616       RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B1,LSB
                GOTO            SADD66L

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK66LL

SADD66L         ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK66LL         RLF             ACC+B1
                RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS1616

                BTFSC           ACC+B1,LSB
                GOTO            SOK66L
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           REM+B0
SOK66L

                endm

UDIV1616L       macro

;       restore = 23 clks,  nonrestore = 17 clks
```


© 1997 Microchip Technology Inc.

```
;       Max Timing:     2+15*23+22 = 369 clks

;       Min Timing:     2+15*17+16 = 273 clks

;       PM: 24                                  DM: 7

                MOVLW           16
                MOVWF           LOOPCOUNT

LOOPU1616       RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0

                BTFSC           _C
                GOTO            UOK66LL
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

                BCF             _C

UOK66LL         RLF             ACC+B1
                RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1616

                endm

UDIV1515L       macro

;       Max Timing:     13+14*18+17+8 = 290 clks

;       Min Timing:     13+14*17+16+3 = 270 clks

;       PM: 42                                  DM: 7

                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B1
                RLF             ACC+B0

                MOVLW           15
                MOVWF           LOOPCOUNT

LOOPU1515       RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
```


© 1997 Microchip Technology Inc.

```
                BTFSS           ACC+B1,LSB
                GOTO            UADD55L

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK55LL

UADD55L         ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK55LL         RLF             ACC+B1
                RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU1515

                BTFSC           ACC+B1,LSB
                GOTO            UOK55L
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           REM+B0
UOK55L

                endm

SDIV1616        macro

;       Max Timing:     7+10+6*14+14+7*14+8 = 221 clks

;       Min Timing:     7+10+6*13+13+7*13+3 = 202 clks

;       PM: 7+10+6*18+18+7*18+8 = 277   DM: 6

                variable i

                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
                RLF             ACC+B0

                i = 2

                while i < 8
```

5

© 1997 Microchip Technology Inc.

```
                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B0,LSB
                GOTO            SADD66#v(i)

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK66#v(i)

SADD66#v(i)     ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK66#v(i)      RLF             ACC+B0

                i=i+1

                endw

                RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B0,LSB
                GOTO            SADD668

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK668

SADD668         ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK668          RLF             ACC+B1

                i = 9

                while i < 16

                RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B1,LSB
                GOTO            SADD66#v(i)

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
```


© 1997 Microchip Technology Inc.

```
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK66#v(i)

SADD66#v(i)     ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK66#v(i)      RLF             ACC+B1

                i=i+1

                endw

                BTFSC           ACC+B1,LSB
                GOTO            SOK66
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           REM+B0
SOK66

                endm

UDIV1616  macro

;       restore = 20 clks,  nonrestore = 14 clks

;       Max Timing: 16*20 = 320 clks

;       Min Timing: 16*14 = 224 clks

;       PM: 16*20 = 320         DM: 6

                variable         i

                i = 0

                while i < 16

                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0

                BTFSC           _C
                GOTO            UOK66#v(i)
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

                BCF             _C

UOK66#v(i)      RLF             ACC+B1
```


© 1997 Microchip Technology Inc.

```
                RLF             ACC+B0


                i=i+1

                endw

                endm

UDIV1515        macro

;       Max Timing:     7+10+6*14+14+7*14+8 = 221 clks

;       Min Timing:     7+10+6*13+13+7*13+3 = 202 clks

;       PM:     7+10+6*18+18+7*18+8 = 277       DM: 6

                variable i

                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
                RLF             ACC+B0

                i = 2

                while i < 8

                RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B0,LSB
                GOTO            UADD55#v(i)

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK55#v(i)

UADD55#v(i)     ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK55#v(i)      RLF             ACC+B0

                i=i+1
```


© 1997 Microchip Technology Inc.

```
                endw

                RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B0,LSB
                GOTO            UADD558

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK558

UADD558         ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK558          RLF             ACC+B1

                i = 9

                while i < 16

                RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W

                BTFSS           ACC+B1,LSB
                GOTO            UADD55#v(i)

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK55#v(i)

UADD55#v(i)     ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK55#v(i)      RLF             ACC+B1

                i=i+1

                endw

                BTFSC           ACC+B1,LSB
                GOTO            UOK55
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           REM+B0
UOK55
```


© 1997 Microchip Technology Inc.

```
                endm

;*****************************************************************************************
;*****************************************************************************************

;       16/16 Bit Signed Fixed Point Divide 16/16 -> 16.16

;       Input:  16 bit fixed point dividend in AARG+B0, AARG+B1
;               16 bit fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD1616S

;       Output: 16 bit fixed point quotient in AARG+B0, AARG+B1
;               16 bit fixed point remainder in REM+B0, REM+B1

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     11+290+3 = 304 clks             A > 0, B > 0
;                               15+290+14 = 319 clks            A > 0, B < 0
;                               15+290+14 = 319 clks            A < 0, B > 0
;                               19+290+3 = 312 clks             A < 0, B < 0

;       Min Timing:     11+255+3 = 269 clks             A > 0, B > 0
;                               15+255+14 = 284 clks            A > 0, B < 0
;                               15+255+14 = 284 clks            A < 0, B > 0
;                               19+255+3 = 277 clks             A < 0, B < 0

;       PM: 19+42+13 = 74               DM: 8

FXD1616S        MOVF            AARG+B0,W
                XORWF           BARG+B0,W
                MOVWF           SIGN
                BTFSS           BARG+B0,MSB         ; if MSB set go & negate BARG
                GOTO            CA1616S

                COMF            BARG+B1
                INCF            BARG+B1
                BTFSC           _Z
                DECF            BARG+B0
                COMF            BARG+B0

CA1616S         BTFSS           AARG+B0,MSB         ; if MSB set go & negate ACCa
                GOTO            C1616S

                COMF            AARG+B1
                INCF            AARG+B1
                BTFSC           _Z
                DECF            AARG+B0
                COMF            AARG+B0

C1616S          CLRF            REM+B0
                CLRF            REM+B1

                SDIV1616L

                BTFSS           SIGN,MSB            ; negate (ACCc,ACCd)
                RETLW           0x00

                COMF            AARG+B1
                INCF            AARG+B1
                BTFSC           _Z
                DECF            AARG+B0
                COMF            AARG+B0

                COMF            REM+B1
```


© 1997 Microchip Technology Inc.

```
                INCF            REM+B1
                BTFSC           _Z
                DECF            REM+B0
                COMF            REM+B0

                RETLW           0x00

;*****************************************************************************************
;*****************************************************************************************

;       16/16 Bit Unsigned Fixed Point Divide 16/16 -> 16.16

;       Input:  16 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               16 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD1616U

;       Output: 16 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               16 bit unsigned fixed point remainder in REM+B0, REM+B1

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     2+369+2 = 373 clks

;       Min Timing:     2+273+2 = 277 clks

;       PM: 2+24+1 = 27         DM: 7

FXD1616U        CLRF            REM+B0
                CLRF            REM+B1

                UDIV1616L

                RETLW           0x00

;*****************************************************************************************
;*****************************************************************************************

;       15/15 Bit Unsigned Fixed Point Divide 15/15 -> 15.15

;       Input:  15 bit unsigned fixed point dividend in AARG+B0, AARG+B1
;               15 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD1515U

;       Output: 15 bit unsigned fixed point quotient in AARG+B0, AARG+B1
;               15 bit unsigned fixed point remainder in REM+B0, REM+B1

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     2+290+2 = 294 clks

;       Min Timing:     2+270+2 = 274 clks

;       PM: 2+42+1 = 45         DM: 7

FXD1515U        CLRF            REM+B0
                CLRF            REM+B1

                UDIV1515L

                RETLW           0x00

                END
;*****************************************************************************************
;*****************************************************************************************
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX Q:32/16 FIXED POINT DIVIDE ROUTINES

```
;       32/16 PIC16 FIXED POINT DIVIDE ROUTINES VERSION 1.5

;       Input:  fixed point arguments in AARG and BARG

;       Output: quotient AARG/BARG followed by remainder in REM

;       All timings are worst case cycle counts

;       It is useful to note that the additional routine FXD3115U
;       can be called in a signed divide application in the special case
;       where AARG > 0 and BARG > 0, thereby offering some improvement in
;       performance.

;         Routine             Clocks     Function

;       FXD3216S    578 32 bit/16 bit -> 32.16 signed fixed point divide

;       FXD3216U    702 32 bit/16 bit -> 32.16 unsigned fixed point divide

;       FXD3115U    541 31 bit/15 bit -> 31.15 unsigned fixed point divide

                list    r=dec,x=on,t=off,p=16C71

                include <PIC16.INC>

;*****************************************************************************************
;*****************************************************************************************

;       Define divide register variables

ACC             equ     0x0D    ; most significant byte of contiguous 4 byte accumulator

SIGN            equ     0x13    ; save location for sign in MSB

TEMP            equ     0x19    ; temporary storage

;       Define binary operation arguments

AARG            equ     0x0D    ; most significant byte of argument A

BARG            equ     0x16    ; most significant byte of argument B

REM             equ     0x11    ; most significant byte of remainder

LOOPCOUNT       equ     0x14    ; loop counter

;       Note:   ( AARG+B0, AARG+B1 )  and  ( ACC+B0, ACC+B1)
;               reference the same storage locations, and similarly for
;               ( REM+B0, REM+B1 ) and ( ACC+B4, ACC+B5 )


;*****************************************************************************************
;*****************************************************************************************

;       32/16 Bit Division Macros

SDIV3216L       macro

;       Max Timing:     9+6*17+16+16+6*17+16+16+6*17+16+16+6*17+16+8 = 537 clks
```


© 1997 Microchip Technology Inc.

```
;       Min Timing:     9+6*16+15+15+6*16+15+15+6*16+15+15+6*16+15+3 = 501 clks

;       PM: 157                                 DM: 9

                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPS3216A      RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B0,LSB
                GOTO            SADD26LA

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK26LA

SADD26LA        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26LA         RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS3216A

                RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B0,LSB
                GOTO            SADD26L8

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK26L8

SADD26L8        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26L8         RLF             ACC+B1

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPS3216B      RLF             ACC+B1,W
```


© 1997 Microchip Technology Inc.

```
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B1,LSB
                GOTO            SADD26LB

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK26LB

SADD26LB        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26LB         RLF             ACC+B1

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS3216B

                RLF             ACC+B2,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B1,LSB
                GOTO            SADD26L16

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK26L16

SADD26L16       ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26L16        RLF             ACC+B2

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPS3216C      RLF             ACC+B2,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B2,LSB
                GOTO            SADD26LC

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK26LC

SADD26LC        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
```


© 1997 Microchip Technology Inc.

```
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26LC         RLF             ACC+B2

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS3216C

                RLF             ACC+B3,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B2,LSB
                GOTO            SADD26L24

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK26L24

SADD26L24       ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26L24        RLF             ACC+B3

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPS3216D      RLF             ACC+B3,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B3,LSB
                GOTO            SADD26LD

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            SOK26LD

SADD26LD        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

SOK26LD         RLF             ACC+B3

                DECFSZ          LOOPCOUNT
                GOTO            LOOPS3216D

                BTFSC           ACC+B3,LSB
                GOTO            SOK26L
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           REM+B0
```


© 1997 Microchip Technology Inc.

```
SOK26L

                endm

UDIV3216L       macro

;       Max Timing:     15+6*22+21+21+6*22+21+21+6*22+21+21+6*22+21+8 = 698 clks

;       Min Timing:     15+6*21+20+20+6*21+20+20+6*21+20+20+6*21+20+3 = 662 clks

;       PM: 233                                 DM: 11

                CLRF            TEMP

                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSS           _C
                INCFSZ          SIGN,W
                SUBWF           TEMP
                RLF             ACC+B0

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU3216A      RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B0,LSB
                GOTO            UADD26LA

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSS           _C
                INCFSZ          SIGN,W
                SUBWF           TEMP
                GOTO            UOK26LA

UADD26LA        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSC           _C
                INCFSZ          SIGN,W
                ADDWF           TEMP

UOK26LA         RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU3216A

                RLF             ACC+B1,W
```


© 1997 Microchip Technology Inc.

```
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B0,LSB
                GOTO            UADD26L8

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSS           _C
                INCFSZ          SIGN,W
                SUBWF           TEMP
                GOTO            UOK26L8

UADD26L8        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSC           _C
                INCFSZ          SIGN,W
                ADDWF           TEMP

UOK26L8         RLF             ACC+B1

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU3216B      RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B1,LSB
                GOTO            UADD26LB

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSS           _C
                INCFSZ          SIGN,W
                SUBWF           TEMP
                GOTO            UOK26LB

UADD26LB        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSC           _C
                INCFSZ          SIGN,W
                ADDWF           TEMP

UOK26LB         RLF             ACC+B1
```


© 1997 Microchip Technology Inc.

```
                DECFSZ          LOOPCOUNT
                GOTO            LOOPU3216B

                RLF             ACC+B2,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B1,LSB
                GOTO            UADD26L16

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSS           _C
                INCFSZ          SIGN,W
                SUBWF           TEMP
                GOTO            UOK26L16

UADD26L16       ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSC           _C
                INCFSZ          SIGN,W
                ADDWF           TEMP

UOK26L16        RLF             ACC+B2

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU3216C      RLF             ACC+B2,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B2,LSB
                GOTO            UADD26LC

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSS           _C
                INCFSZ          SIGN,W
                SUBWF           TEMP
                GOTO            UOK26LC

UADD26LC        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSC           _C
                INCFSZ          SIGN,W
```


© 1997 Microchip Technology Inc.

```
                ADDWF           TEMP

U0K26LC         RLF             ACC+B2

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU3216C

                RLF             ACC+B3,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B2,LSB
                GOTO            UADD26L24

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSS           _C
                INCFSZ          SIGN,W
                SUBWF           TEMP
                GOTO            U0K26L24

UADD26L24       ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSC           _C
                INCFSZ          SIGN,W
                ADDWF           TEMP

U0K26L24        RLF             ACC+B3

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU3216D      RLF             ACC+B3,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B3,LSB
                GOTO            UADD26LD

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                CLRF            SIGN
                CLRW
                BTFSS           _C
                INCFSZ          SIGN,W
                SUBWF           TEMP
                GOTO            U0K26LD

UADD26LD        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0
```


© 1997 Microchip Technology Inc.

```
                CLRF            SIGN
                CLRW
                BTFSC           _C
                INCFSZ          SIGN,W
                ADDWF           TEMP

U0K26LD         RLF             ACC+B3

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU3216D

                BTFSC           ACC+B3,LSB
                GOTO            U0K26L
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           REM+B0
U0K26L

                endm

UDIV3115L       macro

;       Max Timing:     9+6*17+16+16+6*17+16+16+6*17+16+16+6*17+16+8 = 537 clks

;       Min Timing:     9+6*16+15+15+6*16+15+15+6*16+15+15+6*16+15+3 = 501 clks

;       PM: 157                                 DM: 9

                MOVF            BARG+B1,W
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                RLF             ACC+B0

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU3115A      RLF             ACC+B0,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B0,LSB
                GOTO            UADD15LA

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            U0K15LA

UADD15LA        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

U0K15LA         RLF             ACC+B0

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU3115A
```


© 1997 Microchip Technology Inc.

```
                RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B0,LSB
                GOTO            UADD15L8

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK15L8

UADD15L8        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK15L8         RLF             ACC+B1

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU3115B      RLF             ACC+B1,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B1,LSB
                GOTO            UADD15LB

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK15LB

UADD15LB        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK15LB         RLF             ACC+B1

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU3115B

                RLF             ACC+B2,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B1,LSB
                GOTO            UADD15L16

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK15L16

UADD15L16       ADDWF           REM+B1
```


© 1997 Microchip Technology Inc.

```
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

U0K15L16        RLF             ACC+B2

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU3115C      RLF             ACC+B2,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B2,LSB
                GOTO            UADD15LC

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            U0K15LC

UADD15LC        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

U0K15LC         RLF             ACC+B2

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU3115C

                RLF             ACC+B3,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B2,LSB
                GOTO            UADD15L24

                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            U0K15L24

UADD15L24       ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

U0K15L24        RLF             ACC+B3

                MOVLW           7
                MOVWF           LOOPCOUNT

LOOPU3115D      RLF             ACC+B3,W
                RLF             REM+B1
                RLF             REM+B0
                MOVF            BARG+B1,W
                BTFSS           ACC+B3,LSB
                GOTO            UADD15LD
```


© 1997 Microchip Technology Inc.

```
                SUBWF           REM+B1
                MOVF            BARG+B0,W
                BTFSS           _C
                INCFSZ          BARG+B0,W
                SUBWF           REM+B0
                GOTO            UOK15LD

UADD15LD        ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCFSZ          BARG+B0,W
                ADDWF           REM+B0

UOK15LD         RLF             ACC+B3

                DECFSZ          LOOPCOUNT
                GOTO            LOOPU3115D

                BTFSC           ACC+B3,LSB
                GOTO            UOK15L
                MOVF            BARG+B1,W
                ADDWF           REM+B1
                MOVF            BARG+B0,W
                BTFSC           _C
                INCF            BARG+B0,W
                ADDWF           REM+B0
UOK15L

                endm

;*****************************************************************************************
;*****************************************************************************************

;       32/16 Bit Signed Fixed Point Divide 32/16 -> 32.16

;       Input:  32 bit fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               16 bit fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD3216S

;       Output: 32 bit fixed point quotient in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               16 bit fixed point remainder in REM+B0, REM+B1

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     11+537+3 = 551 clks             A > 0, B > 0
;                               15+537+20 = 572 clks            A > 0, B < 0
;                               21+537+20 = 578 clks            A < 0, B > 0
;                               25+537+3 = 565 clks             A < 0, B < 0

;       Min Timing:     11+501+3 = 515 clks             A > 0, B > 0
;                               15+501+20 = 536 clks            A > 0, B < 0
;                               21+501+20 = 542 clks            A < 0, B > 0
;                               25+501+3 = 529 clks             A < 0, B < 0

;       PM: 25+157+19 = 201             DM: 10

FXD3216S        MOVF            AARG+B0,W
                XORWF           BARG+B0,W
                MOVWF           SIGN
                BTFSS           BARG+B0,MSB       ; if MSB set go & negate BARG
                GOTO            CA3216S

                COMF            BARG+B1
                INCF            BARG+B1
```

```
                BTFSC           _Z
                DECF            BARG+B0
                COMF            BARG+B0

CA3216S         BTFSS           AARG+B0,MSB         ; if MSB set go & negate ACCa
                GOTO            C3216S

                COMF            AARG+B3
                INCF            AARG+B3
                BTFSC           _Z
                DECF            AARG+B2
                COMF            AARG+B2
                BTFSC           _Z
                DECF            AARG+B1
                COMF            AARG+B1
                BTFSC           _Z
                DECF            AARG+B0
                COMF            AARG+B0

C3216S          CLRF            REM+B0
                CLRF            REM+B1

                SDIV3216L

                BTFSS           SIGN,MSB        ; negate (ACCc,ACCd)
                RETLW           0x00

                COMF            AARG+B3
                INCF            AARG+B3
                BTFSC           _Z
                DECF            AARG+B2
                COMF            AARG+B2
                BTFSC           _Z
                DECF            AARG+B1
                COMF            AARG+B1
                BTFSC           _Z
                DECF            AARG+B0
                COMF            AARG+B0

                COMF            REM+B1
                INCF            REM+B1
                BTFSC           _Z
                DECF            REM+B0
                COMF            REM+B0

                RETLW           0x00


;*****************************************************************************************
;*****************************************************************************************

;       32/16 Bit Unsigned Fixed Point Divide 32/16 -> 32.16

;       Input:  32 bit unsigned fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               16 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD3216U

;       Output: 32 bit unsigned fixed point quotient in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               16 bit unsigned fixed point remainder in REM+B0, REM+B1

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     2+698+2 = 702 clks

;       Max Timing:     2+662+2 = 666 clks
```


© 1997 Microchip Technology Inc.

```
;       PM: 2+233+1 = 236               DM: 11

FXD3216U        CLRF            REM+B0
                CLRF            REM+B1

                UDIV3216L

                RETLW           0x00

;*****************************************************************************************
;*****************************************************************************************

;       31/15 Bit Unsigned Fixed Point Divide 31/15 -> 31.15

;       Input:  31 bit unsigned fixed point dividend in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               15 bit unsigned fixed point divisor in BARG+B0, BARG+B1

;       Use:    CALL    FXD3115U

;       Output: 31 bit unsigned fixed point quotient in AARG+B0, AARG+B1,AARG+B2,AARG+B3
;               15 bit unsigned fixed point remainder in REM+B0, REM+B1

;       Result: AARG, REM  <-  AARG / BARG

;       Max Timing:     2+537+2 = 541 clks

;       Min Timing:     2+501+2 = 505 clks

;       PM: 2+157+1 = 160               DM: 9

FXD3115U        CLRF            REM+B0
                CLRF            REM+B1

                UDIV3115L

                RETLW           0x00

                END
;*****************************************************************************************
;*****************************************************************************************
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Implementing a Table Read

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

This application note shows how to implement a table look-up for all PICmicro™ products. The examples shown are for the PIC16CXXX family. An explanation of differences between the PIC16C5X family and the PIC17C42 is at the end of this application note.

To access data in program memory, a table read operation must be performed. The table consists of a series of `retlw K` instructions where, the 8-bit table constants are assigned to the literal K. The first instruction in the table computes the offset to the table by using `addwf pcl`, and consequently the program branches to the appropriate `retlw K` instruction (Example 1).

EXAMPLE 1:

```
    .
    .
    movlw   offset  ;load offset in w reg
    call    Table
    .
    .
    .
Table:
    addwf   pcl     ;add offset to pc to
                    ;generate a computed goto
    retlw   'A'     ;return the ASCII char A
    retlw   'B'     ;return the ASCII char B
    retlw   'C'     ;return the ASCII char C
    .
    .
    .
```

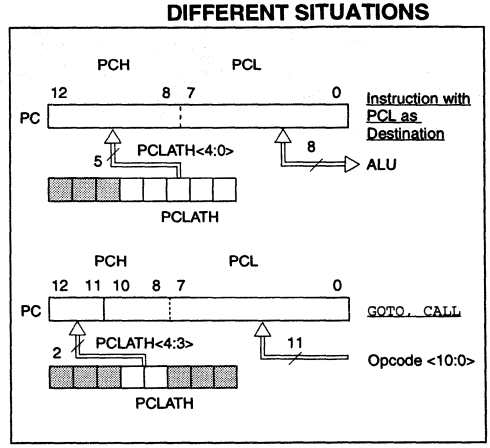
The method is straight forward, however certain precautions have to be exercised when doing a table read in the PIC16CXXX.

IMPLEMENTATION

Program Counter Loading

The Program Counter (PC) in the PIC16CXXX is 13-bits wide. The low 8-bits (PCL) are mapped in RAM at location 02h and are directly readable and writable. The high 5-bits are not accessible directly and can only by written through the PCLATH register (Figure 1). The PCLATH register is a R/W register with only five of its bits implemented <4:0>, all other bits read as '0'.

FIGURE 1: LOADING OF PC IN DIFFERENT SITUATIONS



5

SECTION 1

CALL and GOTO Instructions

When executing a CALL or GOTO, the low 11-bits are loaded directly from the instruction opcode. The high 2-bits are loaded from bits 3 and 4 of the PCLATH register. It is a good practice to pre-load PCLATH with the high byte of the location of the routine before executing the routine. This can be done as follows:

EXAMPLE 2:

```
    .
    .
movlw   HIGH Table  ;load high 8-bit
                    ;address of Table
movwf   PCLATH      ;into PCLATH
call    Routine     ;execute Call
                    ;instruction
    .
    .
```

Note: If the program memory size is less than 2K-words, then the above precaution is not necessary.

Computed GOTO Instruction

Any instruction with PCL as the destination, will load the PCH with the 5 low bits from the PCLATH (Figure 1). In Program 3, if the address where the CALL was made, was on page 0 and the address of the actual table was on page 3, then when executing the computed GOTO, the program will go to a location in page 0 instead of a location on page 3. To avoid the program from branching "erratically" when doing a table read, the PCLATH register should be pre-loaded with the high byte of the "Table" address. Example 3 shows how this can be done.

EXAMPLE 3:

```
    .
    org     0x80    ;code location in page 0
    movlw   offset  ;load offset in w reg
    call    Table
    .
    .
    org     0x0320  ;Table located in page 3
Table:
    addwf   pcl     ;add offset to pc to
                    ;generate a computed goto
    retlw   'A'     ;return the ASCII char A
    retlw   'B'     ;return the ASCII char B
    retlw   'C'     ;return the ASCII char C
    .
    .
    .
```


© 1997 Microchip Technology Inc.

When doing a computed GOTO for a table read, care should taken about page boundaries. The ADDWF PCL instruction will not compute a value greater than 8-bits. In Example 4, the result of the computed GOTO will result in a branch to an unintended portion of the code. The user either has to be cautious as to where in a page the Table is resident or will have to monitor page roll-over and add it to the PCLATH ahead of the computed GOTO.

EXAMPLE 4:

```
    .
    org     0x80        ;code location in
                        ; page 0
    movlw   HIGH Table  ;load PCLATH with hi
                        ; address
    movwf   PCLATH      ;   /
    movlw   offset      ;load offset in w reg
    call    Table
    .
    .
    org     0x02ff      ;Table located end of
                        ; page 2
Table:
    addwf   pcl         ;value in pc will not
                        ; roll over to page 3
    retlw   'A'         ;return the ASCII
                        ; char A
    retlw   'B'         ;return the ASCII
                        ; char B
    retlw   'C'         ;return the ASCII
                        ; char C
    .
    .
    .
```

To take care of both table location and page boundary crossing, it is necessary to do a 13-bit computed GOTO operation as shown in Example 5.

The code in Example 5 will allow the user to place and access a table anywhere in program memory.

EXAMPLE 5:

```
    movlw   LOW Table   ;get low 8 bits of
                        ; address
    addwf   offset      ;do an 8-bit add
                        ; operation
    movlw   HIGH Table  ;get high 5 bits of
                        ; address
    btfsc   status,c    ;page crossed?
    addlw   1           ;yes then increment
                        ; high address
    movwf   PCLATH      ;load high address in
                        ; latch
    movf    offset,w    ;load computed offset
                        ; in w reg
    call    Table
    .
    .
Table:
    movwf   pcl         ;load computed offset
                        ; in PCL
    retlw   'A'         ;return the ASCII
                        ; char A
    retlw   'B'         ;return the ASCII
                        ; char B
    retlw   'C'         ;return the ASCII
                        ; char C
    .
    .
    .
```

5

© 1997 Microchip Technology Inc.

SECTION 2

Implementation for the PIC16C5X Family

The PIC16C5X has no PCH or PCLATH register, so the user has to take into consideration all the precautions mentioned in Section 1. In the PIC16C5X, the location of the Table has to be in the top-half of a 512 word page. This restriction is not valid for the PIC16CXXX family. To convert a table read operation from PIC16C5X code to the PIC16CXXX code, the following should be done:

- Remove any program memory page select instructions (PIC16C56/57), if present.
- Do a 13-bit computed GOTO operation (as shown in Example 5), when doing a table read operation.

Implementation for the PIC17C42

Unlike the PIC16CXXX family, the PIC17C42 loads the PCLATH register with the PCH value during a CALL or GOTO operation, however a computed GOTO, does not take care of page boundary crossing. A 16-bit computed jump address should be calculated before the table read is executed. Example 6 shows how this is done.

Example 6 allows the user to locate the Table at any program memory location, however, for large tables or tables which cross page boundaries, it is recommended that the tablrd/tlrd instructions be used, these instructions are specific for table read operations and are very code efficient.

EXAMPLE 6:

```
    .
    movlw   LOW Table    ;get low 8 bits of
                         ; address
    addwf   offset       ;do an 8-bit add
                         ; operation
    movlw   HIGH Table   ;get high 8 bits of
                         ; address
    btfsc   statist,c    ;page crossed?
    addlw   1            ;yes then increment
                         ; high address
    movwf   PchBuffer,w  ;load in temporary
                         ;location
    call    Table
    .
    .
Table:
    movf    PchBuffer,w  ;get high offset
    movwf   PCLATH       ;load in latch
    movf    offset,w     ;get low offset
    movwf   pcl          ;load computed
                         ; offset in PCL
    retlw   'A'          ;return the ASCII
                         ; char A
    retlw   'B'          ;return the ASCII
                         ; char B
    retlw   'C'          ;return the ASCII
                         ; char C
    .
    .
    .
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN576

Techniques to Disable Global Interrupts

Author: *Mark Palmer*
Microchip Technology Inc.
Contributions by Martin Burghardt
Manager Applications (Central Europe)

INTRODUCTION

This application note discusses four methods for disabling global interrupts. The method best suited for the application may then be used. All discussion will be specific to the PIC16CXXX family of products, but these concepts are also applicable to the PIC17C42, and are shown in the even numbered examples. Note that the PIC17C42's global interrupt bit is called GLINTD and has an inverse sense compared to the GIE bit of the PIC16CXXX family.

To disable all interrupts, either the Global Interrupt Enable (GIE) bit must be cleared or all the individual interrupt enable bits must be cleared. An issue arises when an instruction clears the GIE bit and an interrupt occurs "simultaneously". For example, when a program executes the instruction `BCF INTCON, GIE` (at address PC), there is a possibility that an interrupt will occur during this instruction. If an interrupt occurs during this instruction, the program would complete execution of this instruction, and then immediately branch to the user's interrupt service routine. This occurs because the GIE bit was not clear (disabled) when the interrupt occurred. Normally at the end of the interrupt service routine is the `RETFIE` instruction. This instruction causes the program to return to the instruction at PC + 1, but also sets the GIE bit (enabled). Therefore the GIE bit is not cleared as expected, and unintended program execution may occur.

One method to ensure that the GIE bit is cleared is shown in Example 1 and Example 2, as well as in the PIC16CXXX data sheets. This method tests the state of the GIE bit, after clearing, to ensure that it was not accidentally set in the user's interrupt service routine by the `RETFIE` instruction. If the GIE bit was accidentally set, the program branches back to the instruction that clears the GIE bit.

In this method, the time to ensure that the GIE bit is cleared is indeterminate. Depending on the frequency of the enabled interrupts during this code segment, unexpected delays into the following code segment may occur. For some applications, this may be undesirable. The following three methods address this issue.

5

EXAMPLE 1: CLEARING THE GIE BIT (DISABLING INTERRUPTS, METHOD 1, PIC16CXXX)

```
LOOP    BCF     INTCON, GIE     ; Disable Global Interrupt
        BTFSC   INTCON, GIE     ; Global Interrupt Disabled?
        GOTO    LOOP            ; NO, try again
        :                       ; YES, continue with program flow
        :
        BSF     INTCON, GIE     ; Re-enable Global Interrupt
```

EXAMPLE 2: SETTING THE GLINTD BIT (DISABLING INTERRUPTS, METHOD 1, PIC17C42)

```
        :
LOOP    BSF     CPUSTA, GLINTD ; Disable Global Interrupt
        BTFSS   CPUSTA, GLINTD ; Global Interrupt Disabled?
        GOTO    LOOP           ; NO, try again
        :                      ; YES, continue with program flow
        :
        BCF     CPUSTA, GLINTD ; Re-enable Global Interrupt
```

© 1997 Microchip Technology Inc.

The second method is to disable the individual interrupt enable bits. If it is known which bits are enabled at this point, it can easily be done. Example 3 and Example 4 show the disabling of interrupts, where it is known which sources are enabled (some peripheral interrupts and the T0CKI pin interrupt).

This method also requires the same number of instructions for the disabling/enabling of interrupts, as method 1, but requires a knowledge of which individual interrupt enable bits need to be disabled and (more importantly) re-enabled. The major advantage of this method is that it can minimize the time delay entering the code segment which follows the point where interrupts are disabled.

EXAMPLE 3: CLEARING KNOWN INDIVIDUAL INTERRUPT ENABLE BITS (METHOD 2, PIC16CXXX)

```
:
MOVLW   b'10011111'     ; Disable  Peripheral and T0CKI pin interrupts,
ANDWF   INTCON, F       ; All other bits unchanged
:
:
:
MOVLW   b'01100000'     ; Re-enable Peripheral and T0CKI pin interrupts,
IORWF   INTCON, F       ; All other bits unchanged
:
```

EXAMPLE 4: CLEARING KNOWN INDIVIDUAL INTERRUPT ENABLE BITS (METHOD 2, PIC17C42)

```
:
MOVLW   b'11110011'     ; Disable Peripheral and T0CKI pin interrupts,
ANDWF   INTSTA, F       ; All other bits unchanged
:
:
:
MOVLW   b'00001100'     ; Re-enable Peripheral and T0CKI pin interrupts,
IORWF   INTSTA, F       ; All other bits unchanged
:
```


© 1997 Microchip Technology Inc.

Method 3 can be used if the states of the individual interrupt enable bits are unknown. A temporary byte of data RAM is required to store the value of the INTCON register. This method is shown in Example 5 and Example 6.

This method also requires more instructions for the disabling/enabling of interrupts than in method 1 or method 2, and also a byte of data RAM to temporarily store the value of the INTCON register. The major advantage of this method is that it minimizes the time delay into the code segment which follows the point where interrupts are disabled.

EXAMPLE 5: CLEARING THE INDIVIDUAL INTERRUPT ENABLE BITS (METHOD 3, PIC16CXXX)

```
    :
    MOVF    INTCON, W           ; Move the value in INTCON to
    MOVWF   S_INTCON            ; a shadow register
    MOVLW   b'10000111'         ; Disable all individual interrupts,
    ANDWF   INTCON, F           ; All other bits unchanged
    :
    :
    :
    MOVF    S_INTCON, W         ; Restore the INTCON register
    IORWF   INTCON, F           ;
    :
```

EXAMPLE 6: CLEARING THE INDIVIDUAL INTERRUPT ENABLE BITS (METHOD 3, PIC17C42)

```
    :
    MOVPF   INTSTA, S_INTSTA    ; Move the value in INTSTA to a shadow register
    MOVLW   b'11110000'         ; Disable all individual interrupts,
    ANDWF   INTSTA, F           ; All other bits unchanged
    :
    :
    :
    MOVFP   S_INTSTA, W         ; Restore the INTSTA register
    IORWF   INTSTA, F           ;
    :
```


© 1997 Microchip Technology Inc.

The final method is to use a RAM location to "shadow" the value of the GIE bit. This shadow bit can then be used in the interrupt service routine to determine which return instruction to use. That is, either the RETURN or the RETFIE (which enables the GIE bit) instruction. Example 7 and Example 8 show this implementation, which require that a general purpose bit be available to hold the "shadow" GIE value. In these examples, the shadow GIE (S_GIE) bit is contained in the register FLAG_REG. If an interrupt occurs during the clearing of the shadow GIE, the interrupt is responded to. At the end of the interrupt service routine, the shadow GIE bit is cleared so the RETURN instruction is executed. The GIE bit remains disabled and program execution returns to the instruction which tries to clear the GIE bit (disable). No interrupts can occur during this instruction since the GIE bit was not re-enabled after the interrupt service routine.

This method also requires more instructions for the disabling/enabling of interrupts than in method 1 or method 2, and a single bit of data RAM to temporarily store the value of the desired GIE value, and increases the interrupt service routine execution time by one instruction cycle, for most occurrences of interrupts (two cycles worst case). The major advantage of this method is that it minimizes the time delay into the code segment which follows the point where interrupts are disabled. Also, the individual interrupt enable bits need not be modified.

EXAMPLE 7: THE "SHADOW" GIE BIT (METHOD 4, PIC16CXXX)

```
    :
            org     0x004
INT_SERVICE_ROUTINE
    :
    :
    BTFSC   FLAG_REG, S_GIE         ; Is the S_GIE bit enabled?
    RETFIE                          ; YES, the GIE should be enabled
    RETURN                          ; NO, the GIE should be disabled
END_INT_SERVICE_ROUTINE
;
MAIN:
    :
    :
    BCF     FLAG_REG, S_GIE         ; Clear the shadow GIE bit
    BCF     INTCON, GIE             ; Disable interrupts by clearing the GIE bit
    :
    :
    :
    BSF     FLAG_REG, S_GIE         ; Set the shadow GIE bit
    BSF     INTCON, GIE             ; Enable interrupts by setting the GIE bit
    :
    :
    END
```

EXAMPLE 8: THE "SHADOW" GLINTD BIT (METHOD 4, PIC17C42)

```
    :
            org     0x004
INT_SERVICE_ROUTINE
    :
    :
    BTFSS   FLAG_REG, S_GLINTD      ; Is the S_GLINTD bit enabled?
    RETFIE                          ; YES, the GLINTD should be enabled
    RETURN                          ; NO, the GLINTD should be disabled
END_INT_SERVICE_ROUTINE
;
MAIN:
    :
    :
    BSF     FLAG_REG, S_GLINTD      ; Set the shadow GLINTD bit
    BSF     CPUSTA, GLINTD          ; Disable interrupts by setting the GLINTD bit
    :
    :
    :
    BCF     FLAG_REG, S_GLINTD      ; Clear the shadow GLINTD bit
    BCF     CPUSTA, GLINTD          ; Enable interrupts by clearing the GLINTD bit
    :
    :
    END
```


© 1997 Microchip Technology Inc.

CONCLUSIION

In conclusion, different methods exist to ensure that all interrupts are disabled. The requirement(s) of the application determines which of the methods is the best fit. A comparison of the different methods is shown in Table 1.

TABLE 1: COMPARISON OF DIFFERENT METHODS

	Program Memory	Data Memory	Cycle Delay (TCY)	
			Best Case	Worst Case
Method 1	2 words * N	—	2	Indeterminate
Method 2	2 words * N	—	1	1 + TISR
Method 3 - PIC16CXXX	4 words * N	1 byte	3	3 + TISR
- PIC17C42	3 words * N	1 byte	2	2 + TISR
Method 4	2 words * N + 2 words	1 bit	1†	1 + (TISR + 2)

Legend: N - Number of occurrences to disable / re-enable interrupts.
TISR - Time to execute the interrupt service routine.
† This method increases the interrupt service routine time (TISR) by 1 cycle for most occurrences (2 cycles worst case).

5

© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN585

A Real-Time Operating System for PICmicro™ Microcontrollers

Author: *Jerry Farmer*
Myriad Development Company

INTRODUCTION

Ever dream of having a Real-Time Kernel for the PIC16CXXX family of microcontrollers? Or ever wonder what Multitasking or Threads are all about? Then this article is for you. We will explore how to implement all of the features of a large Real-Time Multitasking Kernel in much less space, with more control, and without the large overhead of existing kernels. By planning ahead, and using the techniques outlined here, you can build your own fast, light, powerful, flexible real-time kernel with just the features needed to get the job done.

Included in this article are two large examples: one on the PIC16C54, and the other on the more powerful PIC16C64. A "Remote Alarm" is implemented on the PIC16C54 as an example of a Non-Preemptive Kernel, with two asynchronous serial input sources capable of running up to 19,200 Baud along with seven sensors needing to be debounced as inputs. One more input line is monitored and causes an internal software recount. For output, this example has an LED that shows eight different internal states of the "Remote Alarm", blinking at different rates and different sequences. Last but not least, is an asynchronous serial output capable of running at 38,400 Baud, passing the inputs to the next remote alarm station. Several short and long timers are included to round out the nine cooperating tasks in this example. Please refer to Figure 2 and Appendix B.

The second example is implemented on an PIC16C64 featuring an interrupt driven Semi-Preemptive Kernel. This example has the serial input and output routines of the first example moved into Interrupt Service Routines (ISR) for more speed and accuracy. The interrupt capabilities of the PIC16C64 will be explored, and a Real-Time Multitasking Kernel framework will be developed. Please refer to Figure 5 and Appendix C.

Why do I Need a Real-Time Kernel?

Real-time design techniques allow the engineer/designer to break-up large, complicated problems into smaller simpler tasks or threads. These more manageable units of code allow faster response to important events, while prioritizing the jobs to be done in a structured well-tested format. The kernel does the job of keeping the time, the peace between tasks, and keeping all the tasks' communication flowing. More activities can be performed in the same amount of time by allowing other tasks to work while other tasks are waiting for some event to occur. Smaller code is also the result of using State-Driven techniques because much information is condensed into the state variables and code structure. If you need an example, look at the PIC16C54's "Remote Alarm" code.

What is Multitasking Anyway?

This is the appearance of several tasks working at the same time. Each task thinks that it owns the CPU, but this appearance is controlled by the kernel. Only one task can be running at a time, but there is undone work that can be done by other tasks not blocked. Multitasking is the orchestration of interrupts, events, communication, shared data, and timing to get a job done. Real-Time Programming is just a bunch of ideas, concepts, and techniques that allow us to divide problems into units of code that are based on units of time, or events that drive a task from one state to another.

5

© 1997 Microchip Technology Inc.

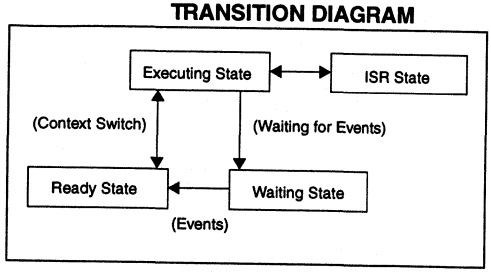
CONCEPTS

We will cover the basic concepts of kernels here so that we are using the same definitions when talking about this difficult topic. This article is a very quick survey on Real-Time Kernel concepts. I hope to get you thinking, reading more, and hopefully writing RT Operating Systems for your current and future projects. Many great books have been written about this very broad and interesting subject. We will refer to some of these books which have a different point of view other than those expressed in this paper.

Critical Section

A critical section is a shared data structure, or a shared resource, or a critical time section of code, or a non-reentrant section of code that can have only one owner that is allowed to view/change/use that section at any one time. These sections must not be interrupted during the update process. They must be protected so that other tasks can not get in and change the pointers/data or modify the hardware at the same time. Remember that if two tasks can get into a critical section, at the same time, then data WILL be corrupted. Make sure that critical sections are small, with time for pending interrupts to get serviced. Not understanding critical sections is where the beginning RT programmers get into the most trouble. Even without interrupts, you must protect variables that are changing over time, such as the byte sized variable `xmt_byte` used in the PIC16C54 example. This variable changes each time the STATE changes for the Serial Out Task. Semaphores, and Disabling Interrupts are two of the techniques used to coordinate between different tasks wanting to control a critical section. Task #4 is devoted to the proper feeding of the shared Serial Out Resource in the PIC16C54 example. Note the use of the binary semaphore "OState_B" to control Task #4, Task #1, and the variable `xmt_byte`. There are several more examples of critical sections in the PIC16C64 example due to the use of interrupts. We disable interrupts for very short time periods to protect these areas. Also in the PIC16C64 example, all critical sections are finished before checking to see if the kernel wants another task to start running instead of the current task. We will discuss in more detail how to protect critical sections later in this article.

FIGURE 1: TASK / PROCESS STATE TRANSITION DIAGRAM



Shared Resources

Data structures, displays, I/O hardware, and non-reentrant routines are a few resource examples. If two or more tasks use these resources, then they are called Shared Resources and you must protect them from being corrupted. They must have only one owner, a way of telling others to wait, and possibly a waiting list for future users of that resource. A rare example of a shared resource is when there exists a critical timing sequence of input and output operations to control some hardware. You must disable interrupts before starting this sequence, and re-enable them upon finishing. Note that Task #1 in the PIC16C64 example is an example of an "non-reentrant" routine that must be finished by the current owner before another task can use it.

Context Switch/Task Switch

When one task takes over from another, the current values of the CPU registers for this running task are saved and the old saved registers for the new task are restored. The new task continues where it left off. This is all done by the Context Switch part of the Real-Time Kernel. Each task usually has a "context switch storage area". Each task's SP (Stack Pointer pointing into its own stack) is stored there along with all the other important saved registers. The "Remote Alarm" example does not need to use a context switch because all the important registers are properly freed-up before each task is finished. The PIC16C64 example uses a similar concept, thus keeping the number of saved registers per task way down. We use an old concept called "where I came from". The variable "FROM" is used to direct the dispatcher to start up the task where it left off. This is because you cannot manipulate the stack in the PIC16CXXX family. This same reason is why we have a "Semi-Preemptive" kernel on the PIC16C64 as an example. By the way, the faster the context switch is done, the better.

Scheduler

The scheduler is that part of the kernel that decides which task will run next. We will talk about several common types in this section. This is where a lot of thinking should be done before starting your new project. By understanding the different kinds of schedulers and what features and problems each type has, you can match your problem to a creatively styled scheduler that meets your needs. For example, the PIC16C54 example shows the recalling of Tasks #1-3 just before a long sequence of code is executed. More creative ways can also be implemented, but be careful to allow all tasks to execute in a timely fashion.

Please see Figure 1. Each task must be in "Ready State" or the "Executing State" to be considered by the scheduler to get temporary control of the CPU next.

© 1997 Microchip Technology Inc.

Non-Preemptive Kernel

The Non-Preemptive Kernel is also called a "Cooperative Kernel" because the tasks only give-up control when they want/need to in coordination with other tasks, and events. The "Remote Alarm" example uses a Non-Preemptive Kernel type, showing that despite its reputation as being a simple kernel type, a lot can be done with it. The Non-Preemptive Kernel type is well suited for the non-interrupt type PIC16C5Xs. The heart beat of the PIC16C54 example is the internal TMR0 counter crossing over from a high value to a low value of the counter. Use the prescaler to adjust the time units. The very fast tasks continually read the TMR0 directly comparing the delta of time to see if it should fire.

Preemptive Kernel

In a Preemptive Kernel, a running task can be swapped out for a higher priority task when it becomes ready. The Preemptive Kernel relies much more on interrupts as its driving force. The context switch is at the heart of this type of kernel. To implement a true Preemptive Kernel, you must be able to manipulate the stack. This is why we implemented a "Semi-Preemptive" kernel on the PIC16C64, with some of the best features of both types of kernels. We moved some of the tasks in the PIC16C54 example into ISRs to handle the I/Os. This works very well as the ISRs are very short and do most of the real work in this example. The TIMER0 interrupt is the heart beat for the PIC16C64 example. You must have a clock interrupt in order to make a true Preemptive kernel.

Round Robin Scheduler

When the scheduler finds tasks on the ready queue that have the same priorities, the scheduler often uses a technique called Round Robin scheduling to make sure each task gets its day in the sun. This means more housekeeping to get it right. This is part of the creative ways you can tailor the scheduler to fit your needs. In the PIC16C54 example, all tasks will get to run shortly after their appointed time. This means that no task will dominate all others in this simple approach. In the "olden" days of the first Real-Time Operating Systems the term was used to mean the same as "time slicing". The Preemptive Kernels of today are a major step forward, with their priority schemes, and intertask communication capabilities.

Preemptive vs. Non-Preemptive

The Preemptive Kernel is harder to develop, but is easier to use, and is sometimes used incorrectly. You must spend more upfront time with the Non-Preemptive Kernel but it is better for more cramped microcontrollers. You get much better response time between a cause/event and the response/action for that event with a Non-Preemptive Kernel. The Preemptive Kernel is more predictable in the response times, and can be calculated as to the maximum time to complete a given job. Often the Preemptive Kernel is more expensive.

Reentrancy

In a Preemptive Kernel, two or more tasks may want to use the same subroutine. The problem is that you can not control when a task is swapped out and when another takes its place. Thus, if a subroutine uses only local or passed variables that are stored only in each tasks' stack, then it is call reentrant or a pure routine. No global variables or hardware may be used in such a pure routine. A way around this reentrancy requirement is to treat the whole subroutine as a critical section.

Appendix D is an example of reentrant code segment as might have been used in the PIC16C54 code example.

Task Priority

Some tasks are not created equal. Some jobs must be done on time or data will be lost. Make the tasks that must get done the highest priority and go down the scale from there. Some kernels make you have a different priority for each task. This is a good idea and requires some thought before coding to make the design work.

Static vs. Dynamic Priorities and Priority Inversions

For most embedded Real-Time Kernels, both static priorities and static tasks are used. Dynamic priorities are sometimes used to solve deadlock and other complex situations that arise from not understanding the problem and not understanding Real-Time Techniques. If the need for dynamic priorities seem to occur, you should relook at how you divided the problem, and divide less so as to include the resources in question under one semaphore. You could divide the problem more to have more tasks not needing two or more resources to complete its job, and have the new tasks talk more together.

© 1997 Microchip Technology Inc.

As for Dynamic tasks, you should define the problem so as to know, ahead of coding, the continuous use of all tasks. You will need more upfront time in the planning stage to get all tasks talking, but it is well worth it to keep Dynamic Priorities and Dynamic Tasking out of the kernel design.

Priority Inversions is a trick used to get past a poorly designed system by inverting the priorities to allow lower tasks to run that were previously blocked. This is a cheap trick, and is best kept out of a Real-Time Kernel. Use the other techniques outlined in this section to solve this kind of problem.

Semaphores

There are basically two types: binary and counting semaphores. The binary semaphore allows only one owner, and all other tasks wanting access are made to wait. The counting semaphore keeps a list of users that need access. Semaphores can be used in many ways. We will illustrate most of them in the following paragraphs. Note that you can implement counting semaphores using binary semaphores.

Mutual Exclusion

We have touched on the subject of Mutual Exclusion earlier (a method to exclude other tasks from gaining access to critical sections). Mutual Exclusion is the process of excluding others from access to the shared resources. To make a semaphore is a very complicated process. The semaphore's construction must be atomic. That means that once the process has started, it can not be interrupted until it has saved the name of the new owner. From there on, it knows that no one else can break-in and change owners. We have implemented a binary semaphore using bits and kernel functions to mutually exclude access in the PIC16C54 example.

In the PIC16C64 example, we also disable interrupts to get the same effect. There are at least two good ways of implementing a binary semaphore. The first and oldest way was discovered by a Dutch mathematician named Dekker. We will refer you to a book that talks more about this algorithm. The second method of implementing a binary semaphore was also discovered by another Dutchman named Dijkstra. This method uses the "testandset" type instruction and is much more important to us. We used the `dec & jump if not zero` instruction (see PIC16C64 example).

Deadlock

Deadlock is a condition where two or more tasks own resources that other tasks need to complete there assignment but will not release their own resources until the other tasks release theirs. Talk about cooperation. Please read section, "Static vs. Dynamic Priorities and Priority Inversions" for a discussion about such problems and ways to solve them. The root of such problems is not understanding the original problem.

Synchronization

Semaphores can be used to synchronize tasks so that messages can be passed between them. Also tasks can be started up by semaphores, stopped by semaphores, or started together. They are the foundation blocks for Real-Time Programming. Once you have built a binary semaphore for your kernel, you can build very complex semaphores to synchronize anything. In the PIC16C54 example, data from several sources are passed out the Serial Port Resource. Task #4 synchronizes the other tasks trying to send data out and synchronizes with task #1 to get it done. When task #1 is running, then task #4 can not run until task #1 is ready for more data to send out.

Intertask Communication

We have touched on this topic already, but for large kernels, one can include more complex communication methods to pass data/messages between tasks. Much of the handshaking is done for you inside the kernel. This takes a lot more space and execution speed to implement them in a kernel.

Event Flags

We implemented Event Flags as simple bits having two states (on and off). More info can be stored per Event Flag such as time it was recorded, by who, and who the event belongs to, and whether data was lost.

Message Mailboxes

This is a nice feature to have if you have the ram space. Mailboxes allow the designer to pass messages between tasks, and allows messages to be looked at when the task is ready, and to reply telling the sender that the message was received. One message can be sent to many tasks at the same time.

Message Queues

This again is a very nice feature if you have the execution time, and the ram to implement them. This feature is related to Mailboxes, in that you can store several messages even after reading, to be processed later. If you want to only operate on the highest prioritized messages before handling the rest, this is allowed. You can be very fancy with the Mailboxes and Queues. If you have them, use them.

© 1997 Microchip Technology Inc.

Interrupts

Interrupts are one of the best inventions to come along for solving Real-Time problems. You can get very quick response to the need, and then go back to what you were doing. The only problem is that they can and do happen at the worst times. That means that you must learn how to turn them on and off to protect your critical sections. Note that before an interrupt can be handled, you must save all important registers so that you can restore them so that the kernel can restart the task where it left off. This is much like the context switch issue, but for interrupts, you must always save and restore. In the PIC16C64 example, the Status, W, and FSR registers are saved in RAM because of the interrupt. The PC register is saved onto the stack by hardware.

Interrupt Latency, Response and Recovery

Interrupt Latency is defined as the largest time period that interrupts are disabled, plus the time it takes for the ISR to start to execute.

The Interrupt Response Time is defined for a Non-Preemptive system as Interrupt Latency plus the "context saving time." For a Preemptive system, add the execution time for the kernel to record the interrupt.

Interrupt Recovery Time for a Non-Preemptive system is defined as the time to restore the saved context and for the restarting of the task that was interrupted. Interrupt Recovery Time for a Preemptive system is the same as for the Non-Preemptive system plus the time the kernel takes in the scheduler deciding which task to run next. These measurements are how most kernels are compared with each other. The PIC16C64 example does very well in these measurements. That is because of the PIC16CXXX processor and that they are mostly a Non-Preemptive system. You must keep the time you disable interrupts to a minimum in any kernel you write or any task that you write. You should break-up long sequences of instructions to allow for interrupts that are already waiting to execute.

ISR Processing Time

ISR (Interrupt Service Routine) Processing Time is defined as the time an ISR keeps control of the CPU. This amount of time should be short, and if a lot of processing needs to be done in a ISR, then break up the ISR. The new ISR should now just store the new data and return. Next, create a new task and move the extra code from the old ISR into the new task. Remember that the longer you are in one interrupt, the longer you can not answer another pressing interrupt.

Nesting interrupts are where the interrupt with a higher priority can interrupt a lower priority interrupt. Care must be used, as different interrupts may have critical sections too, and disabling interrupts must be used here too to protect critical sections. Nesting of interrupts may not exist on all microcontrollers, such as the PIC16CXXX family.

Non-Maskable Interrupts

On some microprocessors, you can enable/disable selected interrupts, such as on the PICmicro family. This is a great tool to control the flow of data into the system and out. Some systems have what is called Non-Maskable Interrupts. Here you can not turn them off by software masking. These NMIs as they are call for short, are used as clock Ticks, because you do not want problems with complex critical sections on a interrupt that you can not turn off. The PIC16CXXX family does not have any NMIs. NMIs are not as useful as maskable interrupts.

Clock Tick

The Clock Tick, is the heart beat of the system. This is how the kernel keeps time (relative & absolute). This is how the kernel is restarted to see if there is a delay that has finished, so that the task can be moved into the ready state. In the PIC16C54 example, the Timer0 clock is used. In the PIC16C64 example, Timer0 is used. You must have a clock interrupt in order to make a true Preemptive kernel. This is the other reason why we implemented a Non-Preemptive Kernel on the PIC16C54 - no clock interrupt.

© 1997 Microchip Technology Inc.

ANALYSIS OF CODE EXAMPLES

These sections are the real meat of this article. In these sections we will explain how the concepts are put to practical use line by line in each of the two main examples - PIC16C54 (Appendix C) and PIC16C64 (Appendix D).

We will also examine a short reentrant code example in Appendix B. We will give some ideas on how to expand the examples and how far and how fast the examples can be pushed. Be sure to read both sections on the two examples.

The "Remote Alarm" application has many interesting features. The concept is to have as many tiers of units like a tree feeding into the lower level units the status of each of the larger branches to one central point. Each unit can detect any changes in status before the intruder shuts that unit down, or tampers with it. If any unit's power or wires connecting it down the tree are cut, the lack of the flow of status and passwords would be noticed in five seconds and reported down the line. The two Serial Input lines per unit receive the status and passwords from it's two larger branches, checking the data and passing the info down the line by its own Serial Output line. The seven input status lines are debounced in these examples, showing the technique.

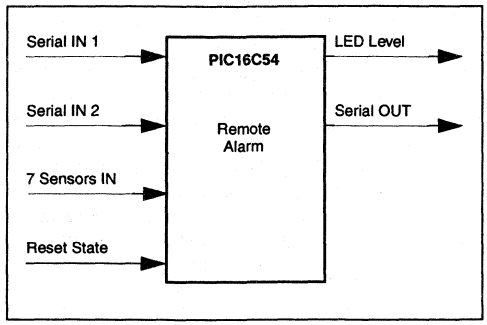
The LED on each unit reports the status at that node as to the importance of its own seven input status lines and the status flowing down the line. The level indication outputted on the LED continues at the last highest level until either a reset is received on the "Reset State" line or five minutes of no new activity on the seven input status lines are received. When either of these two events occur, the level of the LED output is adjusted to the current level of input. Some of the features are changed for this article (Figure 2 and Figure 5).

Another Embedded System use of this type of "Remote Alarm" application is that of placing the unit on the outside of a safe. Hopefully the intruder would be detected before arriving at the unit itself. The continuous stream of status and passwords to the larger unit inside would slow down any simple theft.

PIC16C54 - "Remote Alarm" Example

This example is a cross between a true application and an attempt to show new concepts and some extra features for show. Some of the application specific code has been removed to show more clearly the possibilities of a Real-time Operating System on the PICmicro family. We chose the Baud rate for the Serial output to be twice the speed of the two Serial inputs because it is harder to accurately output a precise Serial Output than it is to monitor Serial inputs.

FIGURE 2: REMOTE ALARM-PIC16C54 EXAMPLE



This example operates at 4 Mhz. By simply increasing the crystal speed to 8 MHz, the two Asynchronous input Serial Baud rates increase from 4800 Baud to 9600 Baud. The Serial Output Baud rate increases from 9600 Baud to 19,200 Baud. By increasing the crystal speed to 16 MHz, it will increase the Baud rates to 19,200 Baud for the two independent Asynchronous inputs, and increase the baud rate for the Asynchronous Serial output to 38,400 Baud. By adjusting the constants in the code for the Serial routines, other Baud rates can be achieved at other crystal speeds. Note that you must use a very stable crystal setup and NOT an RC combination to run these examples.

We will now give a quick outline of the PIC16C54 code example. Lines 1-85 are the equates for this program. Lines 88-95 are simple jump tables so as to save some of the precious "first 256 bytes" of each page. The Serial Output Routines - Task #1 are in lines 97-159. Task #7's subroutines start at line 160 and continue to line 277. In this section, the LED output is controlled. The subroutine QCheck_T123, lines 278-301, is used to allow the checking of Tasks #1-3 to see if they are ready to execute before a long section of code in a slower Task is about to be executed. This is a creative way for the Kernel's Scheduler to makes sure that the highest Prioritized Tasks get serviced before the less important tasks get executed. Task #2 starts at line 302. This task reads the Serial Input #1 for Asynchronous data. Task #2 can be described as a State Machine for outputting a byte Serially. Task #3 interrupts the code of Task #2 at line 333 and continues until line 362. Task #3 also reads the Serial Input but on input #2. Task #2's subroutines continue at line 363 and continue until line 423. Task #3's subroutines continue at line 424 and continue until line 484 is reached. The main or starting code is started at line 485. From that line to line 515, all variables are initialized, and all tasks are initialized at this time also. The Main Loop is started at line 516 and ends at line 665. This is where the real action is done. Each task checks the time to see if the conditions are correct for it to run. The tasks that are not Blocked, and have a job to do now are in a Ready State. In the Main Loop, we check the current state of each task in order of Priority (1-9). If ready, we do a very simple Task

© 1997 Microchip Technology Inc.

Switch and place that task in the Executing State/Running State. Several time unit changes take place in the Main Loop. Tasks #1-4 use 2 µs as a time base by reading the TMR0 directly. A time unit change takes place at lines 562-575 to 512 µs per unit for Tasks #5-6. Another time unit change takes place for Tasks #7-9, to 131072 µs per unit. For Tasks #5-9, each task counts the time units and compares them to their standard for activation or activity. Task #4 starts at line 538 and finishes at line 561. Task #4 controls the feeding of Task #1 from several other tasks that want data to be outputted. It uses several Semaphores to make sure that Task #1 is not bothered until it is ready for another byte. Task #5 monitors the Level Reset Line, and is always running. It simply resets the status of the LED, to be recalculated in Task #6. Task #5 runs through lines 576-581, and is very short. Lines 582-611 represent Task #6. Here we debounce the seven sensor input lines, leaving the current standard in the variable "Old_RB". Task #6 asks/Signals Task #4 to output the current standard out the Serial pin. Task #7's main code is lines 621-628. Task #8 is a five second lack of activity timer, and exists in lines 629-645. If no data comes from either of the two input Serial lines, then Task #8 Signals Task #4 to send a special byte to be outputted by Task #1. This Signals the next "Remote Alarm" of the lack of communication between units. The last task is Task #9. This is a five minute lack of Severe Errors the from Sensor Reset Timer. Lines 646-663 compose Task #9. Subroutine `Do_D_H_E_L` starts at line 667 and continues through to line 692. This routine determines the Highest Error Level, and passes Task #7, the current state, to output on the LED. Lines 693-703, clear the registers #7-1Fh. The "jump at Power-On" code is the last lines 705-706.

The following sections describe in more detail how and what each part of the code does and why. The code segment lines 1-87 are explained in this paragraph. Line 4 tells the MPASM assembler which PICmicro you are using. The include file `PICREG.H` follows with the equates and assignments to make the code more readable and changeable. You should use equates that relate symbols to each other. The Constants — lines 10-12 are the values to change for different Baud rates. They represent the Bit Times for the Baud rates divided by 2 minus some latency factor. You might have to adjust the "Fudge Factor" and other values to fine tune the performance. The value used for the "Fudge Factor" is related to the longest path of code. Lines 21-24 are an experiment that allows a simple name to be associated to a single bit. This allows for easily changeable assignments. Lines 30-54 are the variable assignments. Variables (lines 35-39) are used as time counters. They count the number of units of time, and are compared to literals to see if an Event has just happened. The bits defined in lines 57-64 are used as Binary Semaphores. They keep Critical Sections of data protected. We will see them in action later in the code. The bits defined in lines 67-73 are error flags. They define the current or last error states of the Serial routines, and whether data was lost coming in or out.

The section of equates in lines 76-85 are used to define the different LED activity. They are used by Task #7 to keep the LED blinking. In lines 89-94, we try to save the all important first 256 bytes of any page.

Task #1 outputs a byte Asynchronously over the Serial Output pin. Task #1 is started at line 98. The time units used for Tasks #1-4 are 2µS. We first sample the TMR0 and store the count. When Tasks #1-4 are then allowed to run, they check the difference between the first sample and the current time. If the delta is greater than or equal to the delay, then that Event has just happened. We first check if the state of the Serial Output is zero. We then jump to OStateS to start the outputting of the "Start Bit". Because any Serial Output timings must be rock solid, we use a trick in lines 101-116 that helps greatly. We check if we are within a certain amount of time BEFORE the deadline and then wait for the time to output another bit. This trick allows us to be within a certain ± amount of time within the expected time to output that bit. With this code, we are about <±8% accurate for the Serial Output. You can only use this trick on the most critical tasks, and only on one. In this section of code, we are constantly checking the delta of time from the "FIRST_TMR0_O" reading and the current reading of TMR0. When we are very close to the output time, we jump to line 117. If we are not even close to the proper time, we exit back to the main loop, so we can check the other timers and tasks. Now look at Figure 4 for a description of the Output Pulses, the "Bit units of Time", and the associated state numbers. Note that the activities are spread out over time.

The timer Events help to define the different states and their associated output activities. Each Event is handled in a very short, well-defined set of code as Task #1. Lines 117-131, are a quick state jump table. You need to break all Real-Time code into very short segments — in and then out. Each segment is just a few lines long. You do your activity, save status, and increment to the next state. Notice that OState0_7 code is used several times to output all 8 bits. The state variable is used also to count the number of bits already outputted. The time to the next outputting of a bit is calculated and is adjusted to take out the accumulation of errors in lines 151-152. We make sure of a full "Stop Bit" length in the OStateE code. In the OStateL code, we reset the OState variable to zero, and tell the world that we are not outputting now in line 157. This is important because we use that bit (OState_B) to Signal that we need to protect the variable xmt_byte that changes over several states. We also use it to Signal that we are ready for another byte to output. Look at Task #4. See how it uses this Semaphore to full advantage. We have just explained a Critical Segment variable as outlined in the theory sections of this article.

© 1997 Microchip Technology Inc.

Task #2 reads the Serial Input line 1, running at 4800 Baud. The code structure is very similar to that of Task #1 (Figure 3). Notice that there are more states than the Serial Output Task #1. Once the "Start Bit" is detected, we half step into the "Start Bit" to see if it was a "False Start" or not. We then sample and store the incoming bits to form an 8-bit byte just like Task #1. We sample the "Stop Bit" to see if it is a "Frame Error". We delay another 1/2 bit to get to the end of the "Stop Bit" if there was an "Frame Error" before resetting Task #1's state to 0. Otherwise, we reset Task #1's state to 0, and Signal that we are ready for another "Start Bit". The just received byte is stored in variable "RCV_Storage". A check is made to see if we already sent out the last received byte before clobbering the old byte with the new byte.

Task #3 reads the Serial Input line 2, running at 4800 Baud. The code structure is the same as Task #2 Figure 3). The received byte is also put into the same storage variable as Task #2 - "RCV_Storage". When either Task #2 or Task #3 receives a valid byte, Task #8's counter is reset. You can up the Baud rate of Task #2 and 3 if you lower the output Baud rate of Task #1. Note that for reading the Serial Input Lines, you can be off by ±15% for each sampling, but not accumulatively.

Task #4 finds the next buffered byte to send out through Task #1. Task #4 also controls the order of which byte goes first over another less important byte of data. It can be said that Task #1 Blocks Task #4 from running. You can think of the Serial Output Line as a Shared Resource. The use of Semaphores here allow the Synchronization of data and actions.

Task #5 monitors the Level Reset Input Line and will reset the LED state variable if the line ever goes low. This task is always in the Ready State. This task is said to simply "pole the input line" when ever it can.

Task #6 debounces the seven sensor input lines, running every 20 ms. The variable "T_20_mS_CO" is incremented every 512 µs (Clock Tick) and is compared to the count needed to equal 20 ms. If it is time, the subroutine QCheck_T123 is called to see if Tasks #1-3 are in the Ready State. If any of the Tasks #1-3 are ready, they are ran and we then continue with Task #6. We compare the current value of the input Port_B to see if it stayed the same from the last reading 20 ms back. If the two readings are the same, then Port_B is considered to be stable and the possibly new value is placed in the variable "Old_RB" to be outputted by Task #1. The subroutine D_H_E_L is called to determine the new LED state. We then check if Task #1 was too busy to output the last sensor status byte, if so then that error is recorded.

FIGURE 3: SERIAL INPUT STATES vs. TIME DIAGRAM

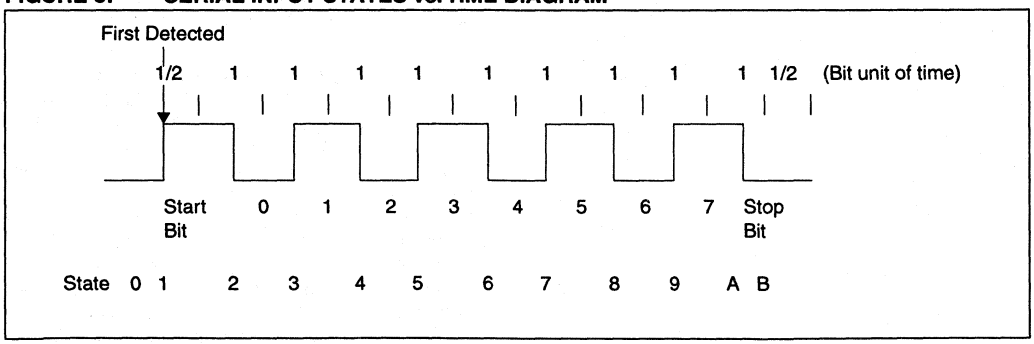
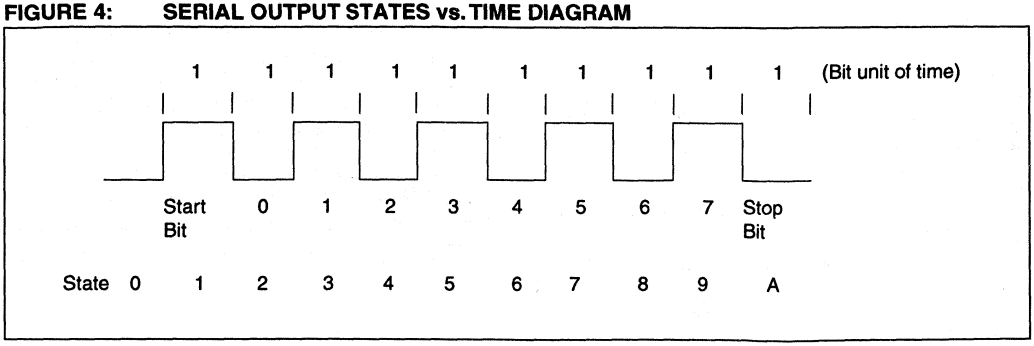


FIGURE 4: SERIAL OUTPUT STATES vs. TIME DIAGRAM



© 1997 Microchip Technology Inc.

Task #7 outputs the Highest Severity Level Indication on the LED. Do_LED starts at line 161, and continues to 276. This task is also broken into small time units of code. It is constantly checking to see if it is time to switch the on/off condition of the LED. The time units for Task #7 are regulated by the code in lines 613-619. 131072 µS = time unit for Tasks #7-9. Task #7 has many state jump tables so it is included in the first 256 bytes of the first page. Lines 168-175 explain the on and off sequences and offs that represent levels of severity of the input status lines. The variable "LED_Mode" has both Task #7's current state number and the sub-state-number for that state's output sequence.

Task #8 is a 5 second lack of input from either of the two Serial input timers. Tasks #2 and #3 will reset the time counter for Task #8, when either receives a full byte. If the time counter "T_5_S_CO" equals 5 secs, then the LED's state is bumped to the highest, and a special byte is sent down the line to the next "Remote Alarm" unit. The counter variable is reset, and count starts all over. We then check if Task #1 was too busy to output the last special status byte, if so then that error is recorded.

Task #9 measures 5 minutes of calm on the 7 sensor lines and then resets the LED's state. Task #9 needs 16 bits of counter power to record 5 minutes of time. The counter variables are reset after being triggered.

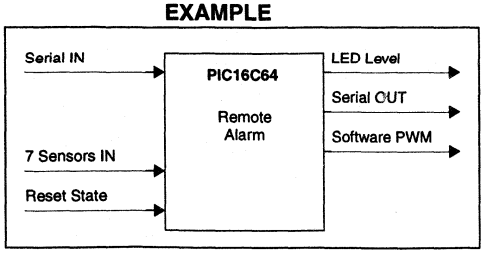
`Do_D_H_E_L` determines the LED's next state based on the 7 sensor input status. This subroutine checks each bit to see if it is active and then checks if a change in the LED's current state needs changing.

`Do_Clear_Regs` clears registers 7-1Fh. It leaves the FSR register zeroed out. This is very important for the PIC16C57 chip.

PIC16C64 - "Remote Alarm64" Example

This example is the same as the PIC16C54 example with a few changes to take advantage of the three timers on the PIC16C64 and interrupts. The second Serial input routine was replaced by an example of a software PWM (Pulse Width Modulation) example. The same code as the PIC16C54 example will run on the PIC16C64 with very few changes using only the TMR0 (TMR0). Be sure to read about the PIC16C54 example, as the comments will not be repeated, except to make a strong point.

FIGURE 5: REMOTE ALARM - PIC16C64 EXAMPLE



This example operates at 4 Mhz. By simply increasing the crystal speeds, you can change the input and output Baud rates just as outlined in the section on the PIC16C54 example's crystal selection. By adjusting the constants in the code for the Serial routines, other Baud rates can be achieved at other crystal speeds.

Note: You must use a very stable crystal setup and NOT an RC combination to run these examples.

We will now give a quick outline of the PIC16C64 code example. Lines 1-78 are the equates for this program. Notice that there is no need for jump tables for subroutines to be in the "first 256 bytes" of each page as there was in the PIC16C54 example. Note that the "Reset Vector" is now at code address 0, and the "Interrupt Vector" is at code address 4. Task #1 and 2 have been simplified greatly by using interrupts and timers. For Task #1, we no longer need to use the "trick" any more. It is time to execute once the routines for Task #1 and others are called. The section of code that handles the "Start Bit" (OStateS) lines 106-122 has been changed to setting up TMR2 with its interrupt to trigger the next call to this subroutine. The initial `CALL` to this subroutine was by Task #4, but later calls are due to Timer 2's interrupts. The amount of time until the next interrupt is set by each state's code. This amount is based on the "Bit Unit of Time" which is based on Baud rate and crystal speed. An easy change to the code is to add a software selectable and "changeable on the fly" Baud rate. This is done by having a variable that selects the new Baud rate from the two data tables. One table gets you the Bit Delay value - see line 110. The other data table gets the value to be put into T2CON - see line 107, which selects the Post and Pre-scalers. You may need to adjust the Bit Delay value to take in account the Interrupt Latency. The OStateL state code shuts down Timer2 and its interrupt. See lines 647-676 to understand how we get here by interrupt. Once Timer 2's count equals the count we put into register PR2, we get an interrupt if the following three conditions are true:

1. Not already in an interrupt. When the current interrupt is done, our interrupt will be executed.
2. GIE and PEIE bits are set.
3. TMR2IE bit is set.

Remember to clear the Flag bit as in line 114 before returning from an interrupt. Return from this subroutine will return you back to Task #4 or back to the ISR handle lines 647-676 depending on who called this routine. The Task #7's subroutines are the same as in the PIC16C54 example, lines 151-268. Task #2 is different from the previous example, lines 288-380. First Task #2 uses two interrupts. The INT interrupt on pin RB0/INT is used to detect the "Start Bit". It is very accurate. It is turned off after the detection in I1StateS code. The second interrupt TMR1 is then Enabled in the I1StateS code. Timer1 is then used to cause an interrupt for all the other states for Task #2. Notice that Timer1 has a

© 1997 Microchip Technology Inc.

16-bit counter and we calculate the amount of Clock Ticks until overflow in lines 329-333. In the state code I1StateL, TMR1 is shut down, and the INT interrupt is now Enabled so as to detect the next input byte. The initializing of the PIC16C64 variable takes place in lines 383-426. The initializing of the tasks take place in lines 427-451. Notice that the last bit to be set is the GIE bit in line 451 after ALL is setup. There are several ways to execute the Task #3-9 code: by Timer0 overflow interrupt, by having the code be in the background as in this example. The trade-offs are many, and too deep for this article. Notice that the subroutine QCheck_T123 is not needed in this method. Timer0 overflow interrupt sets the flag: Time_Bit. The code in lines 454-457 can be considered the "IDLE Task" on some systems. It waits for a Clock Tick from TMR0's overflow. Task #3 is new, and is a simple 8-bit software PWM. Lines 459-478 show how to have 8 bits of ON, and 8 bits of OFF. This task has two states, on and off. You may add to the code by allowing the Real-Time changing of the 8-bit values under software control. When you change the values in the variables PWM_Out and PWM_In, disable all interrupts by using the following line: `BCF INTCON,GIE`, and enable all interrupts by using the following line: `BSF INTCON,GIE`. The new values will be used at the next transition, thus allowing a smooth change. This task could easily be used in the PIC16C54 example type Kernel. Task #4 is the same except that it calls Task #1's subroutine to initiate the outputting of a byte. See line 503. Tasks #5-9 are the same as in the PIC16C54 example. The subroutines: `D_H_E_L` and `Clear_Regs` are the same in both examples. The TMR0 (Timer0) Overflow interrupt ISR (Interrupt Service Routine) is lines 641-645. This ISR will set the Time_Bit bit and clear the Flag that caused the interrupt. The Interrupt code lines 647-676 handles the saving of the Context Registers and the restoring of the Context Registers (W, Status, FSR) and by checking the order which interrupts are to be handled first - see lines 656-669. A very important line is 654. You must set the memory page pointers here for the ISR routines! Line 676 is the only place that an interrupt is allowed to return and set the GIE bit (`RETFIE`).

Reentrant example

See Appendix E for the short code segment. This code corresponds to lines 302-332 in the PIC16C54 example. The purpose of reentrant code is to allow two or more tasks to use the same code at the "same time". See the section about reentrant in the theory section of this article. Notice how the registers 18h-1Bh match the registers 1Ch-1Fh, both starting with the state variable for the two tasks using this routine. Note how Task #2 and Task #3 load a pointer to the state variable for their task before calling DO_I State code. By using the FSR register as a pointer, and incrementing or decrementing the FSR register, you can keep the variables in the two tasks straight even if the two tasks are using different code in the subroutine at any one time. This method is not easy to implement, as can be seen, so use two copies for readability instead, like the PIC16C54 example.

SUMMARY

Now that the PICmicro family of microcontrollers have a way of executing Real-Time Programs, using the techniques outlined in this article, there is very little that PICmicros cannot do! Much more than was ever dreamed before. Many of you will quickly understand and start modifying these examples. Great. That means that we have done our job at Myriad. A few of you may want more help. Great. At Myriad Development Co., we LOVE the PICmicro family.

© 1997 Microchip Technology Inc.

BIBLIOGRAPHY

Foster, Caxton C.
Real Time Programming - Neglected Topics
Reading, Massachusetts
Addison-Wesley Publishing Company, 1981

Holt, R.C., Graham, G.S., Lazowska, E.D., Scott, M.A.
Structured CONCURRENT PROGRAMMING with Operating Systems Applications
Reading, Massachusetts
Addison-Wesley Publishing Company, 1978

Kaisler, Stephen H.
The Design of Operating Systems for Small Computer Systems
New York, NY
John Wiley & Sons, 1983

Labrosse, Jean J.
uC/OS - The Real-Time Kernel
Lawrence, Kansas
R & D Publications, 1992

Loeliger, R.G.
Threaded Interpretive Languages
Peterborough, NH
BYTE BOOKS, 1981

© 1997 Microchip Technology Inc.

APPENDIX A:

A Real-Time Vocabulary

ASYNCHRONOUS - An activity that can happen at any moment, at any time.

BLOCKING - The act of wanting to waiting for an EVENT before continuing.

CLOCK TICK - The heart beat that all time is based on.

CONTEXT/TASK SWITCH - Module that saves and restores the states of a task.

CRITICAL SECTION - Section of code or hardware - only one user at a time.

DEADLOCK - That is where two TASKs are waiting for each others resources.

DISPATCHING - The act of starting up a TASK to run from an RT Kernel.

DYNAMIC PRIORITIES - The ability for TASKs to have there PRIORITIES changed.

DYNAMIC TASKING - The creation and the killing of TASKs.

EMBEDDED SYSTEM - An internal system that operates all by itself.

ENABLING/DISABLING INTERRUPTS - Controlling the interrupting processing.

EVENT - Timer, communication, handshaking, interrupts, data, external events.

EVENT FLAGS - The storage of current states or info on what has happened.

INTERRUPT - A hardware event (external/internal) that triggers a jump to the ISR routines to handle that event.

INTERRUPT LATENCY - How long it takes once signaled to start an ISR.

INTERRUPT RECOVERY - How long it takes once interrupted to return back to code.

KERNEL - Module that controls TASKs, INTERRUPTs, and intertask communications.

MAILBOXES - Away to pass data from one TASK to another.

MASKABLE INTERRUPTS - The ability to control whether an ISR is called or not.

MULTITASKING - The act of several TASKs thinking they own the CPU.

MUTUAL EXCLUSION - The act of allowing only ONE owner to a RESOURCE.

NMI - NON-MASKABLE INTERRUPT - Can not be turned off by software.

READY STATE - Referring to a list of TASKs ready (having work to do NOW).

REENTRANT - Code that can be used by several TASKs at the same time.

RESOURCE - Data structures, display, I/O hardware, non-reentrant routines.

RUNNING STATE - Referring to the ONE task owning/using the CPU currently .

SCHEDULER - That part of a kernel that decides which TASK to run next.

SEMAPHORES - A protocol to control RESOURCES, SIGNAL EVENTS, synchronize tasks.

SIGNAL - The act of one task signaling another that something has happened.

STATE MACHINE - An important concept in dividing a job into TASKs & ISRs.

SYNCHRONIZATION - Were TASKs synchronize over data or at a special time.

TASK PRIORITY - Each TASK is ranked as to its importance to getting done.

TASK/THREAD - Code that is defined by a small coherent job/work to be done.

TIME SLICING - The act of giving the same amount of "time" to each TASK to run.

TRAP - A software caused interrupt, useful for system access.

WAITING STATE - Referring to a list of TASKs waiting for an EVENT(s).

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code.
Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe®
(CompuServe membership not required).

APPENDIX B:

```
MPASM 01.40 Released            APP_B.ASM   1-16-1997  17:09:04         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         list    p=16C54,t=ON,c=132
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;
                     00005 ; 'Reentrant Code Example' Designed by Myriad Development Co. - Jerry
Farmer
                     00006 ;       PIC16C54, 4MHz Crystal, WatchDog Timer OFF
                     00007 ;
                     00008 ;       Program:          APP_B.ASM
                     00009 ;       Revision Date:
                     00010 ;                         1-15-97   Compatibility with MPASMWIN 1.40
                     00011 ;
                     00012 ;*****************************************************************
                     00013 ;
                     00014 ; Register Files
  00000018           00015 IState1         equ     18h   ;Serial In #1 State
  00000019           00016 First_TMR0_I1   equ     19h   ;Starting time for next #1 Input event
  0000001A           00017 nbti1           equ     1Ah   ;Next Bit #1 In Time - variable time
  0000001B           00018 rcv_byte_1      equ     1Bh   ;Receive Serial #1 In byte
  0000001C           00019 IState2         equ     1Ch   ;Serial In #2 State
  0000001D           00020 First_TMR0_I2   equ     1Dh   ;Starting time for next #2 Input event
  0000001E           00021 nbti2           equ     1Eh   ;Next Bit #2 In Time - variable time
  0000001F           00022 rcv_byte_2      equ     1Fh   ;Receive Serial #2 In byte
                     00023
                     00024     INCLUDE    <P16C5X.INC>
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File,Version 3.30 Microchip Technology,Inc.
                     00224         LIST
                     00025
  00000007           00026 temp          EQU     07h   ;Temporary holding register - PIC16C54/56
  00000010           00027 IStateS       EQU     10H
  00000011           00028 IStateS2      EQU     11H
  00000012           00029 IState0_7     EQU     12H
  00000013           00030 IStateE       EQU     13H
  00000014           00031 IStateL       EQU     14H
                     00032
                     00033 ;******  ;Task 2,3 - Asynchronous 2400 Baud Serial Input (LOW=0)
0000                 00034 Do_IState
0000 0220            00035               movf    INDF, F   ;if IState2 == 0
0001 0643            00036               btfsc   STATUS,Z  ;  then Do Start Bit
0002 0A10            00037               goto    IStateS
0003 0201            00038               movf    TMR0,W    ;Get current time
0004 0027            00039               movwf   temp
0005 02A4            00040               incf    FSR, F    ;Point to First_TMR0_I(1,2)
0006 0200            00041               movf    INDF,W    ;Get elapsed time; Time Unit = 2 uS
0007 00A7            00042               subwf   temp, F
0008 02A4            00043               incf    FSR, F    ;Point to nbti(1,2)
0009 0200            00044               movf    INDF,W    ;Past time for next input bit ?
000A 0087            00045               subwf   temp,W
000B 0703            00046               btfss   STATUS,0
000C 0A1E            00047               goto    L1
000D                 00048 L0
```

```
000D 0C02           00049               movlw   2          ;Point to IState(1,2)
000E 00A4           00050               subwf   FSR, F
000F 0200           00051               movf    INDF,W     ;Get (0-B) mode #
0010 0E0F           00052               andlw   H'0F'      ;Get only mode #
0011 01E2           00053               addwf   PCL, F     ;jump to subroutine
                    00054
0012 0A10           00055               goto    IStateS    ;Serial Start Bit
0013 0A11           00056               goto    IStateS    ;1/2 of Start Bit - see if False Start
0014 0A12           00057               goto    IState0_7  ;Bit 0
0015 0A12           00058               goto    IState0_7  ;Bit 1
0016 0A12           00059               goto    IState0_7  ;Bit 2
0017 0A12           00060               goto    IState0_7  ;Bit 3
0018 0A12           00061               goto    IState0_7  ;Bit 4
0019 0A12           00062               goto    IState0_7  ;Bit 5
001A 0A12           00063               goto    IState0_7  ;Bit 6
001B 0A12           00064               goto    IState0_7  ;Bit 7
001C 0A13           00065               goto    IStateE    ;Serial Stop Bit
001D 0A14           00066               goto    IStateL    ;Last State
001E                00067 L1
001E 0064           00068               clrf    FSR        ;Clear the FSR register
001F 0800           00069               retlw   0
                    00070
                    00071 ;*****
0020                00072 Task_2       ;Task 2 - Asynchronous 2400 Baud Serial Input (LOW=0)
0020 0C18           00073               movlw   IState1     ;Point to IState1
0021 0024           00074               movwf   FSR
0022 0900           00075               call    Do_IState
                    00076 ;*****
0023                00077 Task_3       ;Task 3 - Asynchronous 2400 Baud Serial Input (LOW=0)
0023 0C1C           00078               movlw   IState2     ;Point to IState2
0024 0024           00079               movwf   FSR
0025 0900           00080               call    Do_IState
                    00081
                    00082                 END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)


0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXX---------- ----------------

All other memory blocks unused.

Program Memory Words Used:    38
Program Memory Words Free:   474


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C:

```
MPASM 01.40 Released            APP_C.ASM   1-16-1997  17:09:32         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;'Remote Alarm' V1.02
                     00002 ;   Designed by Myriad Development Co/- Jerry Farmer
                     00003 ;     PIC16C54, 4MHz Crystal,
                     00004 ;       WatchDog Timer OFF, MPASM instruction set
                     00005 ;
                     00006 ;       Program:          APP_C.ASM
                     00007 ;       Revision Date:
                     00008 ;                         1-15-97  Compatibility with MPASMWIN 1.40
                     00009 ;
                     00010 ;*****************************************************************
                     00011 ;
                     00012         list    p=16C54,t=ON,c=132
                     00013
                     00014         include "P16C5X.INC"
                     00001         LIST
                     00002 ;P16C5X.INC Standard Header File, Ver. 3.30 Microchip Technology,Inc.
                     00224         LIST
                     00015
                     00016 ; Constants
00000000             00017 INDIR          equ  0          ;Indirect Register
00000033             00018 OUT_BIT_TIME   equ  33h        ;9600 Baud, 104uS Bit Rate
00000064             00019 IN_BIT_TIME    equ  64h        ;4800 Baud, 208uS Bit Rate
00000023             00020 FUDGE_TIME     equ  23h        ;Current Time within a Fudge Factor
                     00021
                     00022 ; B Register Definitions
                     00023 #define Level_Reset     PORTB,  ;Low will cause Past Level to reset
                     00024                                   ;RB.7 - RB.1 == Input from Sensors
000000FF             00025 RB_TRIS        equ  B'11111111'  ;RB TRIS at INIT State == all input
00000000             00026 RB_MASK        equ  B'00000000'  ;What is High/Low for RB at INIT State
                     00027
                     00028 ; A Register Definitions - Programmable Inputs
                     00029 #define Serial_IN_1     PORTA,0  ;Serial Input #1 - 8 bits
                     00030 #define LED             PORTA,1  ;LED Output - Level/State Indicator
                     00031 #define Serial_Out      PORTA,2  ;Serial Output - 8 bits + passwords
                     00032 #define Serial_IN_2     PORTA,3  ;Serial Input #2 - 8 bits
                     00033
000000F9             00034 RA_TRIS        equ  B'11111001' ;RA TRIS at INIT State
00000000             00035 RA_MASK        equ  B'00000000' ;What is High/Low for RA at INIT State
                     00036
                     00037 ; Register Files
00000007             00038 temp           equ   07h       ;Temporary holding register - PIC16C54/56
00000008             00039 Timer_Bits     equ   08h       ;Indicates which Timer(s) are Active = 1
00000009             00040 Flags          equ   09h       ;Error Flags
0000000A             00041 LED_Mode       equ   0Ah       ;(0-2)=Mode, 3=LED_B, (4-6)=Seq #, 7=NEW
0000000B             00042 OState         equ   0Bh       ;Serial Out State
0000000C             00043 T_5_M_LO       equ   0Ch       ;5 Min Timer Counter - Low
0000000D             00044 T_5_M_HI       equ   0Dh       ;5 Min Timer Counter - High
0000000E             00045 T_5_S_CO       equ   0Eh       ;5 Second Timer - lack of Serial Input
0000000F             00046 T_20_mS_CO     equ   0Fh       ;20 mS Timer - used for debouncing
00000010             00047 LED_C          equ   10h       ;LED Counter
00000011             00048 Last_TMR0      equ   11h       ;Last value of the TMR0
00000012             00049 First_TMR0_O   equ   12h       ;Starting time for next Output event
00000013             00050 xmt_byte       equ   13h       ;Serial xmit byte - destroyed in use
00000014             00051 cc             equ   14h       ;256 * TMR0 time
```


© 1997 Microchip Technology Inc.

```
00000015          00052 RCV_Storage   equ   15h       ;Long term storage of rcv_byte #1 & 2
00000016          00053 Old_RB        equ   16h       ;Oldest/Master copy of RB
00000017          00054 Last_RB       equ   17h       ;Last copy of RB
00000018          00055 IState1       equ   18h       ;Serial In #1 State
00000019          00056 First_TMR0_I1 equ   19h       ;Starting time for next #1 Input event
0000001A          00057 nbti1         equ   1Ah       ;Next Bit #1 In Time - variable time
0000001B          00058 rcv_byte_1    equ   1Bh       ;Receive Serial #1 In byte
0000001C          00059 IState2       equ   1Ch       ;Serial In #2 State
0000001D          00060 First_TMR0_I2 equ   1Dh       ;Starting time for next #2 Input event
0000001E          00061 nbti2         equ   1Eh       ;Next Bit #2 In Time - variable time
0000001F          00062 rcv_byte_2    equ   1Fh       ;Receive Serial #2 In byte
                  00063
                  00064 ; Indicates which Timer(s) are Active = 1 & Flags
                  00065 #define OState_B      Timer_Bits,0;Serial Out Active Bit
                  00066 #define IState1_B     Timer_Bits,1;Serial IN #1 Active Bit
                  00067 #define IState2_B     Timer_Bits,2;Serial IN #2 Active Bit
                  00068 #define T_5_S_B       Timer_Bits,3;5 Second Timer Active Bit
                  00069 #define T_5_M_B       Timer_Bits,4;5 Min Timer Active Bit
                  00070 #define RCV_Got_One_B Timer_Bits,5;Got a NEW Received byte to send out
                  00071 #define RB_NEW_B      Timer_Bits,6;Indicates a change in RB input
                  00072 #define S_5_S_B       Timer_Bits,7;Serial In 5 secs of inactivity
                  00073
                  00074 ; Error Flags
                  00075 #define FS_Flag_1      Flags,0    ;Serial #1 IN had a False Start Error
                  00076 #define FE_Flag_1      Flags,1    ;Last Serial #1 IN had a Frame Error
                  00077 #define FS_Flag_2      Flags,2    ;Serial #2 IN had a False Start Error
                  00078 #define FE_Flag_2      Flags,3    ;Last Serial #2 IN had a Frame Error
                  00079 #define RCV_Overflow   Flags,4    ;Lost Serial Input Byte - too Slow
                  00080 #define RB_Overflow    Flags,5    ;Lost RB Input Byte - too Slow
                  00081 #define S_5_S_Overflow Flags,6    ;Lost '5S Inactivity' msg - too Slow
                  00082
                  00083 ;Equates for LED Task #7
                  00084 #define LED_B          LED_Mode,3     ;LED is active
                  00085 #define LED_NEW_B      LED_Mode,7     ;LED has just changed Modes = 1
00000008          00086 LED_OFF_MODE    equ    B'00001000'    ;LED OFF
00000089          00087 LED_SEQ1_MODE   equ    B'10001001'    ;LED Sequence 1: .2s On, 1s Off
0000008A          00088 LED_SEQ2_MODE   equ    B'10001010'    ;LED Sequence 2: 3x(.2s), 1s Off
0000008B          00089 LED_SEQ3_MODE   equ    B'10001011'    ;LED Sequence 3: 5x(.2s), 1s Off
0000009C          00090 LED_SLOW_MODE   equ    B'10011100'    ;LED Slow Pulsing - .3 Hz
0000009D          00091 LED_MEDIUM_MODE equ    B'10011101'    ;LED Medium Pulsing - 1 Hz
0000009E          00092 LED_FAST_MODE   equ    B'10011110'    ;LED Fast Pulsing - 3 Hz
0000008F          00093 LED_ON_MODE     equ    B'10001111'    ;LED ON Continuously
                  00094
                  00095
                  00096 ; Clear Registers 7-1Fh
0000              00097 Clear_Regs
0000 0BE9         00098         GOTO    Do_Clear_Regs   ;Save space in first 256 bytes
                  00099
                  00100 ; Determine the Highest Error Level & Start Task #7 outputing the new
0001              00101 D_H_E_L
0001 0BD2         00102         GOTO    Do_D_H_E_L      ;Save space in first 256 bytes
                  00103                                 ;Level
                  00104 ;******         ;Task #1 - Asynchronous 9600 Baud Serial Output (LOW=0)
0002              00105 Do_OState
0002 022B         00106      MOVF    OState, F       ;if OState == 0
0003 0643         00107      BTFSC   STATUS,Z        ;
0004 0A24         00108      GOTO    OStateS         ;then goto Output-Start-Bit
0005 0201         00109      MOVF    TMR0,W          ;Get current time
0006 0027         00110      MOVWF   temp            ; & store in Temporary variable
0007 0212         00111      MOVF    First_TMR0_O,W  ;Get elapsed time; Time Unit = 2 uS
0008 00A7         00112      SUBWF   temp, F         ;Delta of Current Time & Orginal Time
0009 0C23         00113      MOVLW   FUDGE_TIME      ;Take in account processing time to do it
000A 0087         00114      SUBWF   temp,W          ;Time within fudge factor ?
000B 0703         00115      BTFSS   STATUS,C
000C 0A23         00116      GOTO    _0005           ;Not time yet to change States so return
000D 0C33         00117 _0003   MOVLW   OUT_BIT_TIME    ;Past time for next out-bit ?
```


© 1997 Microchip Technology Inc.

```
000E 0087           00118          SUBWF   temp,W
000F 0603           00119          BTFSC   STATUS,C        ;Do some delaying until it is time
0010 0A15           00120          GOTO    _0004           ;It is now time to out put a bit
0011 0C04           00121          MOVLW   H'04'           ;Account for loop delay
0012 01E7           00122          ADDWF   temp, F
0013 0000           00123          NOP                     ;   make loop delay even
0014 0A0D           00124          GOTO    _0003           ;Wait for exact time to output bit
0015 020B           00125 _0004    MOVF    OState,W        ;Get (0-A) mode #
0016 0E0F           00126          ANDLW   H'0F'           ;Get only mode #
0017 01E2           00127          ADDWF   PCL, F          ;jump to subroutine
0018 0A24           00128          GOTO    OStateS         ;Serial Start Bit
0019 0A2B           00129          GOTO    OState0_7       ;Bit 0
001A 0A2B           00130          GOTO    OState0_7       ;Bit 1
001B 0A2B           00131          GOTO    OState0_7       ;Bit 2
001C 0A2B           00132          GOTO    OState0_7       ;Bit 3
001D 0A2B           00133          GOTO    OState0_7       ;Bit 4
001E 0A2B           00134          GOTO    OState0_7       ;Bit 5
001F 0A2B           00135          GOTO    OState0_7       ;Bit 6
0020 0A2B           00136          GOTO    OState0_7       ;Bit 7
0021 0A31           00137          GOTO    OStateE         ;Serial Stop Bit
0022 0A36           00138          GOTO    OStateL         ;Last State
0023 0800           00139 _0005    RETLW   H'00'
                    00140
0024                00141 OStateS
0024 0545           00142          BSF     Serial_Out      ;Serial Start Bit
0025 0201           00143          MOVF    TMR0,W          ;Store starting time
0026 0032           00144          MOVWF   First_TMR0_O
0027 0C0D           00145          MOVLW   H'0D'           ;Fudge again
0028 00B2           00146          SUBWF   First_TMR0_O, F
0029 02AB           00147          INCF    OState, F       ;increment to next state
002A 0800           00148          RETLW   H'00'
                    00149
002B                00150 OState0_7                        ;Bit 0 - 7
002B 0333           00151          RRF     xmt_byte, F     ;Move bit into C from right most bit
002C 0703           00152          BTFSS   STATUS,C        ;
002D 0445           00153          BCF     Serial_Out      ;
002E 0603           00154          BTFSC   STATUS,C        ;
002F 0545           00155          BSF     Serial_Out      ;
0030 0A32           00156          GOTO    OS_End
0031                00157 OStateE
0031 0445           00158          BCF     Serial_Out      ;Serial Stop Bit
0032 0C33           00159 OS_End   MOVLW   OUT_BIT_TIME    ;Adjust out the cumulation of error
0033 01F2           00160          ADDWF   First_TMR0_O, F
0034 02AB           00161          INCF    OState, F       ;increment to next state
0035 0800           00162          RETLW   H'00'
0036                00163 OStateL
0036 006B           00164          CLRF    OState          ;Ready to send next byte out
0037 0408           00165          BCF     OState_B        ;Serial Out not active
0038 0800           00166          RETLW   H'00'
                    00167
                    00168 ;******          ;Task #7 - Output Highest Level Indication on LED
0039                00169 Do_LED
0039 06EA           00170          BTFSC   LED_NEW_B       ;Initialize regs if change in modes
003A 0A4C           00171          GOTO    LED_NEW
003B 02B0           00172          INCF    LED_C, F        ;Inc Counter - Time Unit = 131072 uS
003C 020A           00173          MOVF    LED_Mode,W      ;Get (0-7) mode #
003D 0E07           00174          ANDLW   H'07'           ;Get only mode #
003E 01E2           00175          ADDWF   PCL, F          ;jump to subroutine
003F 0A48           00176          GOTO    LED_OFF         ;LED OFF
0040 0A64           00177          GOTO    LED_SEQ1        ;LED Seq 1: 1 short pulse & pause
0041 0A67           00178          GOTO    LED_SEQ2        ;LED Seq 2: 2 short pulses & pause
0042 0A8A           00179          GOTO    LED_SEQ3        ;LED Seq 3: 3 short pulses & pause
0043 0A50           00180          GOTO    LED_SLOW        ;LED Slow Pulsing - .3 Hz
0044 0A5E           00181          GOTO    LED_MEDIUM      ;LED Medium Pulsing - 1 Hz
0045 0A61           00182          GOTO    LED_FAST        ;LED Fast Pulsing - 3 Hz
0046 0A4D           00183          GOTO    LED_ON          ;LED ON Continuously
```


© 1997 Microchip Technology Inc.

```
0047 0800          00184 _0012   RETLW   H'00'
                   00185 ;------
0048               00186 LED_OFF
0048 0425          00187         BCF     LED             ;Turn off LED
0049 046A          00188         BCF     LED_B           ;LED must be off
004A 0070          00189         CLRF    LED_C           ;Reset Counter - LED_C = 0
004B 0800          00190         RETLW   H'00'
                   00191 ;------
004C               00192 LED_NEW
004C 04EA          00193         BCF     LED_NEW_B       ;Done initializing
004D               00194 LED_ON
004D 0525          00195         BSF     LED             ;Turn on LED
004E 0070          00196         CLRF    LED_C           ;Reset Counter - LED_C = 0
004F 0800          00197         RETLW   H'00'
                   00198 ;------
0050               00199 LED_SLOW
0050 0C0C          00200         MOVLW   H'0C'           ;.3Hz @ 50% Duty
0051 0027          00201         MOVWF   temp
0052 0207          00202 LED_S   MOVF    temp,W          ;Check LED_C if time, .3Hz @ 50% Duty
0053 0090          00203         SUBWF   LED_C,W
0054 0743          00204         BTFSS   STATUS,Z
0055 0A47          00205         GOTO    _0012
0056 0C10          00206         MOVLW   H'10'
0057 01AA          00207         XORWF   LED_Mode, F     ;Switch states
0058 078A          00208         BTFSS   LED_Mode,4      ;Now make LED same state
0059 0425          00209         BCF     LED
005A 068A          00210         BTFSC   LED_Mode,4
005B 0525          00211         BSF     LED
005C 0070          00212         CLRF    LED_C           ;Reset LED_C
005D 0800          00213         RETLW   H'00'
                   00214 ;------
005E               00215 LED_MEDIUM
005E 0C04          00216         MOVLW   H'04'           ;1Hz @ 50% Duty
005F 0027          00217         MOVWF   temp
0060 0A52          00218         GOTO    LED_S           ;Go do it
                   00219 ;------
0061               00220 LED_FAST
0061 0C01          00221         MOVLW   H'01'           ;3Hz @ 50% Duty
0062 0027          00222         MOVWF   temp
0063 0A52          00223         GOTO    LED_S           ;Go do it
                   00224 ;------
0064               00225 LED_SEQ1                        ;.2 ON, 1 OFF
0064 078A          00226         BTFSS   LED_Mode,4      ;Skip if bit is high
0065 0A76          00227         GOTO    ON1             ;Go do it
0066 0A82          00228         GOTO    OFF3            ;Go do it
                   00229 ;------
0067               00230 LED_SEQ2                        ;.2 ON, .2 OFF, .2 ON, 1 OFF
0067 020A          00231         MOVF    LED_Mode,W
0068 0027          00232         MOVWF   temp
0069 0C30          00233         MOVLW   H'30'           ;Get sequence # only
006A 0167          00234         ANDWF   temp, F
006B 03A7          00235         SWAPF   temp, F         ;swap nibbles
006C 0207          00236         MOVF    temp,W          ;get nibble for offset
006D 01E2          00237         ADDWF   PCL, F          ;Table jump calculation
006E 0A76          00238         GOTO    ON1             ;LED is on, check if time to change
006F 0A7C          00239         GOTO    OFF2            ;LED is off, check if time to change
0070 0A76          00240         GOTO    ON1             ;LED is on, check if time to change
0071 0A82          00241         GOTO    OFF3            ;LED is off, check if time to change
                   00242 ;------
0072               00243 LED_Exit
0072 0C10          00244         MOVLW   H'10'           ;Inc Seq #
0073 01EA          00245         ADDWF   LED_Mode, F
0074 0070          00246         CLRF    LED_C           ;Reset LED_C
0075 0800          00247         RETLW   H'00'
0076               00248 ON1
0076 0C02          00249         MOVLW   H'02'           ;Check LED_C if time, .2 sec-on
```


© 1997 Microchip Technology Inc.

```
0077 0090           00250          SUBWF   LED_C,W
0078 0743           00251          BTFSS   STATUS,Z
0079 0A47           00252          GOTO    _0012
007A 0425           00253          BCF     LED             ;Turn off LED
007B 0A72           00254          GOTO    LED_Exit
007C                00255 OFF2
007C 0C02           00256          MOVLW   H'02'           ;Check LED_C if time, .2 sec-on
007D 0090           00257          SUBWF   LED_C,W
007E 0743           00258          BTFSS   STATUS,Z
007F 0A47           00259          GOTO    _0012
0080 0525           00260          BSF     LED             ;Turn on LED
0081 0A72           00261          GOTO    LED_Exit
0082                00262 OFF3
0082 0C08           00263          MOVLW   H'08'           ;Check LED_C if time, 1 sec-off
0083 0090           00264          SUBWF   LED_C,W
0084 0743           00265          BTFSS   STATUS,Z
0085 0A47           00266          GOTO    _0012
0086 0525           00267          BSF     LED             ;Turn on LED
0087 0CF0           00268          MOVLW   H'F0'
0088 012A           00269          IORWF   LED_Mode, F     ;Cause (Seq# & NEW) to overflow to 0
0089 0A72           00270          GOTO    LED_Exit
008A                00271 LED_SEQ3      ;.2 ON, .2 OFF, .2 ON, .2 OFF, .2 ON, 1 OFF
008A 020A           00272          MOVF    LED_Mode,W      ;Get LED info
008B 0027           00273          MOVWF   temp
008C 0C70           00274          MOVLW   H'70'           ;Get sequence # only
008D 0167           00275          ANDWF   temp, F
008E 03A7           00276          SWAPF   temp, F         ;swap nibbles
008F 0207           00277          MOVF    temp,W          ;get nibble for offset
0090 01E2           00278          ADDWF   PCL, F          ;Table jump calculation
0091 0A76           00279          GOTO    ON1             ;LED is on check if time to change
0092 0A7C           00280          GOTO    OFF2            ;LED is off check if time to change
0093 0A76           00281          GOTO    ON1             ;LED is on check if time to change
0094 0A7C           00282          GOTO    OFF2            ;LED is off check if time to change
0095 0A76           00283          GOTO    ON1             ;LED is on check if time to change
0096 0A82           00284          GOTO    OFF3            ;LED is off check if time to change
                    00285
                    00286 ;****            Quick Check of Tasks #1, #2 and #3
0097                00287 QCheck_T123
                    00288          ;Task #1 - Asynchronous 9600 Baud Serial Output (LOW=0)
0097 0708           00289          BTFSS   OState_B        ;if not outputing now then skip call
0098 0A9A           00290          GOTO    T2
0099 0902           00291          CALL    Do_OState       ;Go Do Task #1
                    00292
                    00293          ;Task #2 - Asynchronous 4800 Baud Serial Input (LOW=0)
009A 0628           00294 T2       BTFSC   IState1_B       ;if already started then call
009B 0A9F           00295          GOTO    _0029
009C 0605           00296          BTFSC   Serial_IN_1     ;if Start bit ? then call
009D 0A9F           00297          GOTO    _0029
009E 0AA0           00298          GOTO    T3
009F 09A7           00299 _0029    CALL    Do_I1State      ;Go Do Task #2
                    00300
                    00301          ;Task #3 - Asynchronous 4800 Baud Serial Input (LOW=0)
00A0 0648           00302 T3       BTFSC   IState2_B       ;if already started then call
00A1 0AA5           00303          GOTO    _0031
00A2 0665           00304          BTFSC   Serial_IN_2     ;if Start bit ? then call
00A3 0AA5           00305          GOTO    _0031
00A4 0800           00306          RETLW   H'00'
00A5 09C2           00307 _0031    CALL    Do_I2State      ;Go Do Task #3
00A6 0800           00308          RETLW   H'00'
                    00309
                    00310 ;******          ;Task #2 - Asynchronous 4800 Baud Serial Input (LOW=0)
00A7                00311 Do_I1State
00A7 0238           00312          MOVF    IState1, F      ;if IState1 == 0
00A8 0643           00313          BTFSC   STATUS,Z        ;               then Do Start Bit
00A9 0ADD           00314          GOTO    I1StateS
00AA 0201           00315          MOVF    TMR0,W          ;Get current time
```


© 1997 Microchip Technology Inc.

```
00AB 0027          00316         MOVWF   temp
00AC 0219          00317         MOVF    First_TMR0_I1,W ;Get elapsed time; Time Unit = 2 uS
00AD 00A7          00318         SUBWF   temp, F
00AE 021A          00319         MOVF    nbti1,W         ;Past time for next input bit ?
00AF 0087          00320         SUBWF   temp,W
00B0 0703          00321         BTFSS   STATUS,C
00B1 0AC1          00322         GOTO    _0033
00B2 0218          00323         MOVF    IState1,W       ;Get (0-B) mode #
00B3 0E0F          00324         ANDLW   H'0F'           ;Get only mode #
00B4 01E2          00325         ADDWF   PCL, F          ;jump to subroutine
00B5 0ADD          00326         GOTO    I1StateS        ;Serial Start Bit
00B6 0AE6          00327         GOTO    I1State2        ;1/2 of Start Bit - see if False Start
00B7 0AEF          00328         GOTO    I1State0_7      ;Bit 0
00B8 0AEF          00329         GOTO    I1State0_7      ;Bit 1
00B9 0AEF          00330         GOTO    I1State0_7      ;Bit 2
00BA 0AEF          00331         GOTO    I1State0_7      ;Bit 3
00BB 0AEF          00332         GOTO    I1State0_7      ;Bit 4
00BC 0AEF          00333         GOTO    I1State0_7      ;Bit 5
00BD 0AEF          00334         GOTO    I1State0_7      ;Bit 6
00BE 0AEF          00335         GOTO    I1State0_7      ;Bit 7
00BF 0AF8          00336         GOTO    I1StateE        ;Serial Stop Bit
00C0 0B03          00337         GOTO    I1StateL        ;Last State - End of Stop Bit
00C1               00338 _0033
00C1 0800          00339         RETLW   H'00'
                   00340
                   00341 ;******          ;Task #3 - Asynchronous 4800 Baud Serial Input (LOW=0)
00C2               00342 Do_I2State
00C2 023C          00343         MOVF    IState2, F      ;if IState1 == 0
00C3 0643          00344         BTFSC   STATUS,Z        ;then Do Start Bit
00C4 0B10          00345         GOTO    I2StateS
00C5 0201          00346         MOVF    TMR0,W          ;Get current time
00C6 0027          00347         MOVWF   temp
00C7 021D          00348         MOVF    First_TMR0_I2,W ;Get elapsed time; Time Unit = 2 uS
00C8 00A7          00349         SUBWF   temp, F
00C9 021E          00350         MOVF    nbti2,W         ;Past time for next input bit ?
00CA 0087          00351         SUBWF   temp,W
00CB 0703          00352         BTFSS   STATUS,C
00CC 0ADC          00353         GOTO    _0035
00CD 021C          00354         MOVF    IState2,W       ;Get (0-B) mode #
00CE 0E0F          00355         ANDLW   H'0F'           ;Get only mode #
00CF 01E2          00356         ADDWF   PCL, F          ;jump to subroutine
00D0 0B10          00357         GOTO    I2StateS        ;Serial Start Bit
00D1 0B19          00358         GOTO    I2StateS2       ;1/2 of Start Bit - see if False Start
00D2 0B22          00359         GOTO    I2State0_7      ;Bit 0
00D3 0B22          00360         GOTO    I2State0_7      ;Bit 1
00D4 0B22          00361         GOTO    I2State0_7      ;Bit 2
00D5 0B22          00362         GOTO    I2State0_7      ;Bit 3
00D6 0B22          00363         GOTO    I2State0_7      ;Bit 4
00D7 0B22          00364         GOTO    I2State0_7      ;Bit 5
00D8 0B22          00365         GOTO    I2State0_7      ;Bit 6
00D9 0B22          00366         GOTO    I2State0_7      ;Bit 7
00DA 0B2B          00367         GOTO    I2StateE        ;Serial Stop Bit
00DB 0B36          00368         GOTO    I2StateL        ;Last State - End of Stop Bit
00DC 0800          00369 _0035   RETLW   H'00'
                   00370
                   00371 ;***             ;Subroutines for Task #2
00DD               00372 I1StateS                        ;Start Bit - Setup timing variables
00DD 0528          00373         BSF     IState1_B       ;Serial Input Active
00DE 0201          00374         MOVF    TMR0,W          ;Store starting time
00DF 0039          00375         MOVWF   First_TMR0_I1
00E0 0C0D          00376         MOVLW   H'0D'           ;Fudge again
00E1 00B9          00377         SUBWF   First_TMR0_I1, F
00E2 0C32          00378         MOVLW   H'32'           ;Time delay = 1/2 bit time
00E3 003A          00379         MOVWF   nbti1
00E4 02B8          00380         INCF    IState1, F      ;Increment to next state
00E5 0800          00381         RETLW   H'00'
```


© 1997 Microchip Technology Inc.

```
00E6                    00382 I1State2                        ;Check if still a Start Bit
00E6 0705               00383         BTFSS   Serial_IN_1    ;False Start Error ?
00E7 0B06               00384         GOTO    FS_Error_1
00E8 0409               00385         BCF     FS_Flag_1       ;Start Bit OK
00E9 021A               00386         MOVF    nbti1,W         ;Adjust out the error
00EA 01F9               00387         ADDWF   First_TMR0_I1, F
00EB 0C64               00388         MOVLW   IN_BIT_TIME     ;Time Delay = full bit time
00EC 003A               00389         MOVWF   nbti1
00ED 02B8               00390         INCF    IState1, F      ;increment to next state
00EE 0800               00391         RETLW   H'00'
00EF                    00392 I1State0_7                      ;Bit 0 - 7
00EF 0705               00393         BTFSS   Serial_IN_1     ;Move Input bit into C
00F0 0403               00394         BCF     STATUS,C
00F1 0605               00395         BTFSC   Serial_IN_1
00F2 0503               00396         BSF     STATUS,C
00F3 033B               00397         RRF     rcv_byte_1, F   ;Move C into left most bit
00F4 021A               00398         MOVF    nbti1,W
00F5 01F9               00399         ADDWF   First_TMR0_I1, F ;Adjust out the error
00F6 02B8               00400         INCF    IState1, F      ;increment to next state
00F7 0800               00401         RETLW   H'00'
00F8                    00402 I1StateE                        ;Check if we have a proper Stop Bit
00F8 0605               00403         BTFSC   Serial_IN_1     ;Frame Error
00F9 0B09               00404         GOTO    F_Error_1
00FA 0429               00405         BCF     FE_Flag_1       ;Stop Bit OK
00FB 006E               00406         CLRF    T_5_S_CO        ;Reset 5 Sec Timer - got a good byte
                        00407         ;Process the msg Here !
00FC 021B               00408         MOVF    rcv_byte_1,W    ;Make a copy of just received byte
00FD 0035               00409         MOVWF   RCV_Storage
00FE 07A8               00410         BTFSS   RCV_Got_One_B   ;Report Lost data
00FF 0489               00411         BCF     RCV_Overflow
0100 06A8               00412         BTFSC   RCV_Got_One_B
0101 0589               00413         BSF     RCV_Overflow
0102 05A8               00414         BSF     RCV_Got_One_B   ;We Now have a RB Value to go out
0103                    00415 I1StateL
0103 0078               00416         CLRF    IState1         ;Ready to receive next byte
0104 0428               00417         BCF     IState1_B       ;Serial In not currently active
0105 0800               00418         RETLW   H'00'
0106                    00419 FS_Error_1                      ;False Start - Shut Down Checking
0106 0428               00420         BCF     IState1_B       ;Serial Input NOT Active
0107 0509               00421         BSF     FS_Flag_1       ;False Start Error
0108 0B03               00422         GOTO    I1StateL        ;Start All Over
0109                    00423 F_Error_1                       ;Frame Error - Wait for End of Stop Bit
0109 021A               00424         MOVF    nbti1,W         ;Adjust out the error
010A 01F9               00425         ADDWF   First_TMR0_I1, F
010B 0C32               00426         MOVLW   H'32'           ;Time Delay = 1/2 bit time
010C 003A               00427         MOVWF   nbti1
010D 0529               00428         BSF     FE_Flag_1       ;Frame Error for this Byte ?
010E 02B8               00429         INCF    IState1, F      ;Increment to next state
010F 0800               00430         RETLW   H'00'
                        00431
                        00432 ;***              ;Subroutines for Task #3
0110                    00433 I2StateS                        ;Start Bit - Setup timing variables
0110 0548               00434         BSF     IState2_B       ;Serial Input Active
0111 0201               00435         MOVF    TMR0,W          ;Store starting time
0112 003D               00436         MOVWF   First_TMR0_I2
0113 0C0D               00437         MOVLW   H'0D'           ;Fudge again
0114 00BD               00438         SUBWF   First_TMR0_I2, F
0115 0C32               00439         MOVLW   H'32'           ;Time delay = 1/2 bit time
0116 003E               00440         MOVWF   nbti2
0117 02BC               00441         INCF    IState2, F      ;Increment to next state
0118 0800               00442         RETLW   H'00'
0119                    00443 I2StateS2                       ;Check if still a Start Bit
0119 0765               00444         BTFSS   Serial_IN_2     ;False Start Error ?
011A 0B39               00445         GOTO    FS_Error_2
011B 0449               00446         BCF     FS_Flag_2       ;Start Bit OK
011C 021E               00447         MOVF    nbti2,W         ;Adjust out the error
```


© 1997 Microchip Technology Inc.

```
011D 01FD           00448          ADDWF   First_TMR0_I2, F
011E 0C64           00449          MOVLW   IN_BIT_TIME     ;Time Delay = full bit time
011F 003E           00450          MOVWF   nbti2
0120 02BC           00451          INCF    IState2, F      ;increment to next state
0121 0800           00452          RETLW   H'00'
0122                00453 I2State0_7                       ;Bit 0 - 7
0122 0765           00454          BTFSS   Serial_IN_2     ;Move Input bit into C
0123 0403           00455          BCF     STATUS,C
0124 0665           00456          BTFSC   Serial_IN_2
0125 0503           00457          BSF     STATUS,C
0126 033F           00458          RRF     rcv_byte_2, F   ;Move C into left most bit
0127 021E           00459          MOVF    nbti2,W
0128 01FD           00460          ADDWF   First_TMR0_I2, F ;Adjust out the error
0129 02BC           00461          INCF    IState2, F       ;increment to next state
012A 0800           00462          RETLW   H'00'
012B                00463 I2StateE                         ;Check if we have a proper Stop Bit
012B 0665           00464          BTFSC   Serial_IN_2     ;Frame Error
012C 0B3C           00465          GOTO    F_Error_2
012D 0469           00466          BCF     FE_Flag_2       ;Stop Bit OK
012E 006E           00467          CLRF    T_5_S_CO        ;Reset 5 Sec Timer - got a good byte
                    00468          ;Process the msg Here !
012F 021F           00469          MOVF    rcv_byte_2,W    ;Make a copy of just received byte
0130 0035           00470          MOVWF   RCV_Storage
0131 07A8           00471          BTFSS   RCV_Got_One_B   ;Report Lost data
0132 0489           00472          BCF     RCV_Overflow
0133 06A8           00473          BTFSC   RCV_Got_One_B
0134 0589           00474          BSF     RCV_Overflow
0135 05A8           00475          BSF     RCV_Got_One_B   ;We Now have a RB Value to go out
0136                00476 I2StateL
0136 007C           00477          CLRF    IState2         ;Ready to receive next byte
0137 0448           00478          BCF     IState2_B       ;Serial In not currently active
0138 0800           00479          RETLW   H'00'
0139                00480 FS_Error_2
0139 0448           00481          BCF     IState2_B       ;False Start - Shut Down Checking
013A 0549           00482          BSF     FS_Flag_2       ;False Start Error
013B 0B36           00483          GOTO    I2StateL        ;Start All Over
013C                00484 F_Error_2                        ;Frame Error - Wait for End of Stop Bit
013C 021E           00485          MOVF    nbti2,W         ;Adjust out the error
013D 01FD           00486          ADDWF   First_TMR0_I2, F
013E 0C32           00487          MOVLW   H'32'           ;Time Delay = 1/2 bit time
013F 003E           00488          MOVWF   nbti2
0140 0569           00489          BSF     FE_Flag_2       ;Frame Error for this Byte ?
0141 02BC           00490          INCF    IState2, F      ;Increment to next state
0142 0800           00491          RETLW   H'00'
                    00492
                    00493 ;******          ;Code Starting point
0143                00494 Main
0143 0C00           00495          MOVLW   H'00'          ;What is High/Low for RA at INIT State
0144 0025           00496          MOVWF   PORTA
0145 0C00           00497          MOVLW   H'00'          ;What is High/Low for RB at INIT State
0146 0026           00498          MOVWF   PORTB
0147 0CF9           00499          MOVLW   H'F9'          ;RA TRIS at INIT State
0148 0005           00500          TRIS    5
0149 0CFF           00501          MOVLW   H'FF'          ;RB TRIS at INIT State
014A 0006           00502          TRIS    6
014B 0C00           00503          MOVLW   H'00'          ;TMR0/2
014C 0002           00504          OPTION
014D 0900           00505          CALL    Clear_Regs     ;Clear Registers 7-1F - Same Memory Page
014E 0061           00506          CLRF    TMR0           ;Start timers
                    00507
                    00508 ;Initialize Tasks
                    00509                                 ;Task #1 waits for byte to output
                    00510                                 ;Task #2 waits for Serial IN Start Bit
                    00511                                 ;Task #3 waits for Serial IN Start Bit
                    00512                                 ;Task #4 runs when Task 1 is Not
                    00513                                 ;Task #5 is always running
```


© 1997 Microchip Technology Inc.

```
014F 0206           00514          MOVF    PORTB,W          ;Task #6 is Initialized here
0150 0036           00515          MOVWF   Old_RB
0151 0216           00516          MOVF    Old_RB,W         ;Make all the same initial value
0152 0037           00517          MOVWF   Last_RB
0153 05C8           00518          BSF     RB_NEW_B         ;Tell Task #4: RB byte ready to output
0154 0C08           00519          MOVLW   LED_OFF_MODE
0155 002A           00520          MOVWF   LED_Mode         ;Task #7 is Started
0156 0568           00521          BSF     T_5_S_B          ;Task #8 is Started here
0157 0588           00522          BSF     T_5_M_B          ;Task #9 is Started here
                    00523
                    00524 ; Handle Task & Timer activities - Main Loop
0158                00525 Task_1   ;Task #1 - Asynchronous 9600 Baud Serial Output (LOW=0)
0158 0708           00526          BTFSS   OState_B         ;if not outputing now then skip call
0159 0B5B           00527          GOTO    Task_2
015A 0902           00528          CALL    Do_OState        ;Go Do Task #1
                    00529
015B                00530 Task_2   ;Task #2 - Asynchronous 4800 Baud Serial Input (LOW=0)
015B 0628           00531          BTFSC   IState1_B        ;if already started then call
015C 0B60           00532          GOTO    _0053
015D 0605           00533          BTFSC   Serial_IN_1      ;if Start bit ? then call
015E 0B60           00534          GOTO    _0053
015F 0B61           00535          GOTO    Task_3
0160 09A7           00536 _0053    CALL    Do_I1State       ;Go Do Task #2
                    00537
0161                00538 Task_3 ;Task #3 - Asynchronous 4800 Baud Serial Input (LOW=0)
0161 0648           00539          BTFSC   IState2_B        ;if already started then call
0162 0B66           00540          GOTO    _0055
0163 0665           00541          BTFSC   Serial_IN_2      ;if Start bit ? then call
0164 0B66           00542          GOTO    _0055
0165 0B67           00543          GOTO    Task_4
0166 09C2           00544 _0055    CALL    Do_I2State       ;Go Do Task #3
                    00545
0167                00546 Task_4   ;Task #4 - Finds next Buffered Byte to Send Out through Task 1
0167 0608           00547          BTFSC   OState_B         ;if outputing now then skip call
0168 0B7D           00548          GOTO    _0059
0169 07A8           00549          BTFSS   RCV_Got_One_B    ;Got a NEW Received byte to send
016A 0B70           00550          GOTO    _0057
016B 0215           00551          MOVF    RCV_Storage,W    ;Send just received byte
016C 0033           00552          MOVWF   xmt_byte
016D 04A8           00553          BCF     RCV_Got_One_B    ;Clear need to send old byte
016E 0508           00554          BSF     OState_B         ;Start Task #1 & Lock Out Others
016F 0B7D           00555          GOTO    _0059
0170 07C8           00556 _0057    BTFSS   RB_NEW_B         ;Indicates a change in RB input
0171 0B77           00557          GOTO    _0058
0172 0216           00558          MOVF    Old_RB,W         ;Send New RB value
0173 0033           00559          MOVWF   xmt_byte
0174 04C8           00560          BCF     RB_NEW_B         ;Clear need to send out newest value
0175 0508           00561          BSF     OState_B         ;Start Task #1 & Lock Out Others
0176 0B7D           00562          GOTO    _0059
0177 07E8           00563 _0058    BTFSS   S_5_S_B          ;Serial In 5 secs of inactivity
0178 0B7D           00564          GOTO    _0059
0179 0CFF           00565          MOVLW   H'FF'            ;Tell of inactivity of Serial In
017A 0033           00566          MOVWF   xmt_byte
017B 04E8           00567          BCF     S_5_S_B          ;Clear need to send msg
017C 0508           00568          BSF     OState_B         ;Start Task #1 & Lock Out Others
                    00569
                    00570         ;Heart Beat - Time unit = 512 uS for Tasks #5 & #6
017D 0201           00571 _0059    MOVF    TMR0,W           ;Step-up time units * 512
017E 0027           00572          MOVWF   temp
017F 0211           00573          MOVF    Last_TMR0,W      ;Test to see if it overflowed
0180 0087           00574          SUBWF   temp,W
0181 0703           00575          BTFSS   STATUS,C
0182 0B86           00576          GOTO    Inc_Time
0183 0207           00577          MOVF    temp,W           ;unit error = < |+-512 uS|
0184 0031           00578          MOVWF   Last_TMR0
0185 0B58           00579          GOTO    Task_1
```


© 1997 Microchip Technology Inc.

```
0186                00580 Inc_Time
0186 0207           00581         MOVF    temp,W          ;Save current TMR0 into Last_TMR0
0187 0031           00582         MOVWF   Last_TMR0
                    00583
0188                00584 Task_5  ;Task #5 - Monitor Level Reset Input Line - Always Running !
0188 0606           00585         BTFSC   Level_Reset
0189 0B8C           00586         GOTO    Task_6
018A 0C08           00587         MOVLW   LED_OFF_MODE    ;Lowest Level Indicator output
018B 002A           00588         MOVWF   LED_Mode
                    00589
018C                00590 Task_6  ;Task #6 - Debounce 8 bit Input Sensors - Runs every 20 mS
018C 02AF           00591         INCF    T_20_mS_CO, F   ;Inc Counter - Time Unit = 512 uS
018D 0C27           00592         MOVLW   H'27'           ;Used to debounce the input
018E 008F           00593         SUBWF   T_20_mS_CO,W
018F 0743           00594         BTFSS   STATUS,Z
0190 0BA7           00595         GOTO    _0065
0191 006F           00596         CLRF    T_20_mS_CO      ;Reset T_20_mS_CO to start over again
                    00597
0192 0997           00598         CALL    QCheck_T123     ;Quick Check of Tasks #1, #2 and #3
                    00599
0193 0206           00600         MOVF    PORTB,W         ;Last copy of RB same as Current ?
0194 0097           00601         SUBWF   Last_RB,W
0195 0643           00602         BTFSC   STATUS,Z
0196 0B9A           00603         GOTO    _0062
0197 0206           00604         MOVF    PORTB,W         ;Store Current RB - diff from Last
0198 0037           00605         MOVWF   Last_RB
0199 0B9C           00606         GOTO    _0063
019A 0217           00607 _0062   MOVF    Last_RB,W       ;New Old RB <- same value over 20 mS
019B 0036           00608         MOVWF   Old_RB
019C 0236           00609 _0063   MOVF    Old_RB, F       ;See if RB is now 0
019D 0643           00610         BTFSC   STATUS,Z        ;RB == 0 ? then keep timer running
019E 0BA1           00611         GOTO    _0064
019F 006C           00612         CLRF    T_5_M_LO        ;Reset 5 Min Timer
01A0 006D           00613         CLRF    T_5_M_HI        ;  still not zero yet
01A1 0901           00614 _0064   CALL    D_H_E_L         ;Determine the Highest Error Level
01A2 07C8           00615         BTFSS   RB_NEW_B        ;Check for Lost Data Error
01A3 04A9           00616         BCF     RB_Overflow
01A4 06C8           00617         BTFSC   RB_NEW_B
01A5 05A9           00618         BSF     RB_Overflow
01A6 05C8           00619         BSF     RB_NEW_B        ;Every 20 mS send Old_RB out
                    00620
                    00621        ;Heart Beat - Time unit = 131072 uS for Tasks #7, #8 & #9
01A7 0CF9           00622 _0065   MOVLW   H'F9'           ;RA TRIS - refresh
01A8 0005           00623         TRIS    5
01A9 0CFF           00624         MOVLW   H'FF'           ;RB TRIS - refresh
01AA 0006           00625         TRIS    6
01AB 02F4           00626         DECFSZ  cc, F           ;Step-up time units * 256
01AC 0B58           00627         GOTO    Task_1
                    00628
01AD                00629 Task_7 ;Task 7 - Output Highest Level Indication on LED
01AD 076A           00630         BTFSS   LED_B           ;Is LED active ?
01AE 0BB1           00631         GOTO    Task_8
                    00632
01AF 0997           00633         CALL    QCheck_T123     ;Quick Check of Tasks #1, #2 and #3
                    00634
01B0 0939           00635         CALL    Do_LED          ;Handle LED timing
                    00636
01B1                00637 Task_8 ;Task #8 - 5 Second Serial Input Lack of Activity Timer
01B1 0768           00638         BTFSS   T_5_S_B         ;5 Sec Timer Active ?
01B2 0BC0           00639         GOTO    Task_9
01B3 02AE           00640         INCF    T_5_S_CO, F     ;Inc Counter - Time Unit = 131072 uS
01B4 0C26           00641         MOVLW   H'26'           ;Check T_5_S_CO if time
01B5 008E           00642         SUBWF   T_5_S_CO,W
01B6 0743           00643         BTFSS   STATUS,Z
01B7 0BC0           00644         GOTO    Task_9
01B8 006E           00645         CLRF    T_5_S_CO        ;Reset T_5_S_CO
```


© 1997 Microchip Technology Inc.

```
01B9 0C8F          00646          MOVLW   LED_ON_MODE      ;Highest Level Indicator output
01BA 002A          00647          MOVWF   LED_Mode
01BB 07E8          00648          BTFSS   S_5_S_B          ;Check if Lost Data Error
01BC 04C9          00649          BCF     S_5_S_Overflow
01BD 06E8          00650          BTFSC   S_5_S_B
01BE 05C9          00651          BSF     S_5_S_Overflow
01BF 05E8          00652          BSF     S_5_S_B          ;Send notice of 5 seconds of inaction
                   00653
01C0               00654 Task_9 ;Task #9 - 5 Min. Lack of Severe Error from Sensors Reset Timer
01C0 0788          00655          BTFSS   T_5_M_B          ;5 Min Timer Active ?
01C1 0BD1          00656          GOTO    Task_A
01C2 02AC          00657          INCF    T_5_M_LO, F      ;Inc LO Counter; Time Unit = 131072 uS
01C3 0643          00658          BTFSC   STATUS,Z         ;See if carry needs to be passed on ?
01C4 02AD          00659          INCF    T_5_M_HI, F      ;Inc HI Counter; Time Unit = 131072 uS
01C5 0C08          00660          MOVLW   H'08'            ;#2288<  Check T_5_M_HI if time
01C6 008D          00661          SUBWF   T_5_M_HI,W
01C7 0743          00662          BTFSS   STATUS,Z
01C8 0BD1          00663          GOTO    Task_A
01C9 0CF0          00664          MOVLW   H'F0'            ;#2288>  Check T_5_M_LO if time
01CA 008C          00665          SUBWF   T_5_M_LO,W
01CB 0743          00666          BTFSS   STATUS,Z
01CC 0BD1          00667          GOTO    Task_A
01CD 006C          00668          CLRF    T_5_M_LO         ;Reset T_5_M_LO
01CE 006D          00669          CLRF    T_5_M_HI         ;Reset T_5_M_HI
01CF 0C08          00670          MOVLW   LED_OFF_MODE     ;Lowest Level Indicator output
01D0 002A          00671          MOVWF   LED_Mode
01D1               00672 Task_A
01D1 0B58          00673          GOTO    Task_1           ;Loop Forever
                   00674
                   00675 ;****
01D2               00676 Do_D_H_E_L       ; Determine the Highest Error Level & Start Task #7
01D2 0C07          00677          MOVLW   H'07'            ;Check top 7 bits
01D3 0027          00678          MOVWF   temp
01D4 0216          00679          MOVF    Old_RB,W         ;Get copy of 7 debounced Sensor Input
01D5 0037          00680          MOVWF   Last_RB
01D6 0377          00681 _0070    RLF     Last_RB, F       ;Put top bit into C bit
01D7 0603          00682          BTFSC   STATUS,C         ;Check if C bit is set
01D8 0BDE          00683          GOTO    _0072
01D9 02E7          00684          DECFSZ  temp, F          ;Continue to check lesser bits
01DA 0BD6          00685          GOTO    _0070
01DB 0206          00686 _0071    MOVF    PORTB,W          ;Restore current value of RB
01DC 0037          00687          MOVWF   Last_RB
01DD 0800          00688          RETLW   H'00'
01DE 020A          00689 _0072    MOVF    LED_Mode,W       ;Get current Level Indicator
01DF 0E07          00690          ANDLW   H'07'            ;Get only       "       "
01E0 0037          00691          MOVWF   Last_RB          ;Store into a temporary register
01E1 0207          00692          MOVF    temp,W           ;Check if already at this Level
01E2 0097          00693          SUBWF   Last_RB,W
01E3 0603          00694          BTFSC   STATUS,C
01E4 0BDB          00695          GOTO    _0071
01E5 0C88          00696          MOVLW   H'88'            ;Start to build LED_Mode
01E6 0107          00697          IORWF   temp,W           ;Put new Level Indicator into reg
01E7 002A          00698          MOVWF   LED_Mode         ;Store new LED Mode
01E8 0BDB          00699          GOTO    _0071
                   00700
01E9               00701 Do_Clear_Regs    ; Clear Registers 7-1Fh
01E9 0C1F          00702          MOVLW   H'1F'            ;First regs to clear
01EA 0024          00703          MOVWF   FSR
01EB 0060          00704 Loop_C   CLRF    INDIR            ;Clear reg
01EC 00E4          00705          DECF    FSR, F           ;point to next reg to clear
01ED 0CE7          00706          MOVLW   H'E7'            ;Dec temp, jump if not done
01EE 0084          00707          SUBWF   FSR,W
01EF 0603          00708          BTFSC   STATUS,C
01F0 0BEB          00709          GOTO    Loop_C
01F1 0064          00710          CLRF    FSR              ;Lastly clear FSR reg
01F2 0800          00711          RETLW   H'00'
```


© 1997 Microchip Technology Inc.

```
                    00712
01FF                00713          ORG     H'1FF'          ;RESET to Main
01FF 0B43           00714          GOTO    Main
                    00715
                    00716          END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXX------------X

All other memory blocks unused.

Program Memory Words Used:   500
Program Memory Words Free:    12


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX D:

```
MPASM 01.40 Released            APP_D.ASM   1-16-1997  17:10:05         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         list    p=16C64,t=ON,c=132
                     00002 ;
                     00003 ;*****************************************************************
                     00004 ;
                     00005 ; 'Remote Alarm64' V1.00
                     00006 ;   Designed by Myriad Development Co. - Jerry Farmer
                     00007 ;   PIC16C64, 4MHz Crystal, WatchDog Timer OFF, MPASM instruction set
                     00008 ;   PIC16C54, 4MHz Crystal, WatchDog Timer OFF
                     00009 ;       Program:          APP_D.ASM
                     00010 ;       Revision Date:
                     00011 ;                         1-15-97   Compatibility with MPASMWIN 1.40
                     00012 ;
                     00013 ;*****************************************************************
                     00014 ;
                     00015
                     00016         include "P16C64.INC"
                     00001         LIST
                     00002 ;P16C64.INC Standard Header File,Ver. 1.01 Microchip Technology, Inc.
                     00238         LIST
                     00017
                     00018 ; B Register Definitions
00000000             00019 Serial_IN_1    equ  0           ;Serial Input #1 - 8 bits - INT pin
                     00020                                 ;RB.7 - RB.1 == Input from Sensors
000000FF             00021 RB_TRIS        equ  B'11111111' ;RB TRIS at INIT State == all input
00000000             00022 RB_MASK        equ  B'00000000' ;What is High/Low for RB at INIT State
                     00023
                     00024 ; A Register Definitions - Programmable Inputs
00000000             00025 Level_Reset    equ  0       ;PORTA.0 - Reset Level Indicator
00000001             00026 LED            equ  1       ;LED Output - Level/State Indicator
00000002             00027 Serial_Out     equ  2       ;Serial Output - 8 bits + passwords
00000003             00028 PWM_Out        equ  3       ;PWM Output - 8 bits ON, 8 bits OFF
                     00029
000000F1             00030 RA_TRIS        equ  B'11110001'  ;RA TRIS at INIT State
00000000             00031 RA_MASK        equ  B'00000000' ;What is High/Low for RA at INIT State
                     00032
                     00033 ; Register Files
00000020             00034 temp           equ  20h  ;Temporary holding register - PIC16C54/56
00000021             00035 tmp            equ  21h  ;Temporary reg
00000022             00036 Temp_W         equ  22h  ;Interrupt storage of W
00000023             00037 Temp_Stat      equ  23h  ;Interrupt storage of STATUS
00000024             00038 Temp_FSR       equ  24h  ;Interrupt storage of FSR
00000025             00039 T_B            equ  25h  ;Indicates which Timer(s) are Active = 1
00000026             00040 FLAGS          equ  26h  ;Error Flags
00000027             00041 LED_Mode       equ  27h  ;(0-2)=Mode, 3=LED_B, (4-6)=Seq #, 7=NEW
00000028             00042 OState         equ  28h  ;Serial Out State
00000029             00043 IState1        equ  29h  ;Serial In #1 State
0000002A             00044 cc             equ  2Ah  ;256 * TMR0 time
0000002B             00045 T_5_M_LO       equ  2Bh  ;5 Min Timer Counter - Low
0000002C             00046 T_5_M_HI       equ  2Ch  ;5 Min Timer Counter - High
0000002D             00047 T_5_S_CO       equ  2Dh  ;5 Second Timer - lack of Serial Input
0000002E             00048 T_20_mS_CO     equ  2Eh  ;20 mS Timer - used for debouncing
0000002F             00049 T_PWM_CO       equ  2Fh  ;PWM Counter
00000030             00050 LED_C          equ  30h  ;LED Counter
00000031             00051 xmt_byte       equ  31h  ;Serial xmit byte - destroyed in use
```

```
00000032          00052 rcv_byte_1      equ  32h  ;Receive Serial #1 In byte
00000033          00053 RCV_Storage     equ  33h  ;Long term storage of rcv_byte #1
00000034          00054 Old_RB          equ  34h  ;Oldest/Master copy of RB
00000035          00055 Last_RB         equ  35h  ;Last copy of RB
00000036          00056 PWM_ON          equ  36h  ;PWM ON Counter
00000037          00057 PWM_OFF         equ  37h  ;PWM OFF Counter
00000038          00058 PWM_tmp         equ  38h  ;PWM temporary counter
                  00059
                  00060 ; Indicates which Timer(s) are Active = 1 & Flags - T_B
00000000          00061 OState_B        equ  0    ;Serial Out Active Bit
00000001          00062 IState1_B       equ  1    ;Serial IN #1 Active Bit
00000002          00063 T_5_S_B         equ  2    ;5 Second Timer Active Bit
00000003          00064 T_5_M_B         equ  3    ;5 Min Timer Active Bit
00000004          00065 RCV_Got_One_B   equ  4    ;Got a NEW Received byte to send out
00000005          00066 RB_NEW_B        equ  5    ;Indicates a change in RB input
00000006          00067 S_5_S_B         equ  6    ;Serial In 5 secs of inactivity
00000007          00068 T_PWM_B         equ  7    ;PWM Activity Bit
                  00069
                  00070 ; Error Flags - FLAGS
00000000          00071 FS_Flag_1       equ  0    ;Serial #1 IN had a False Start Error
00000001          00072 FE_Flag_1       equ  1    ;Last Serial #1 IN had a Frame Error
00000002          00073 RCV_Overflow    equ  2    ;Lost Serial Input Byte - too Slow
00000003          00074 RB_Overflow     equ  3    ;Lost RB Input Byte - too Slow
00000004          00075 S_5_S_Overflow  equ  4    ;Lost '5S Inactivity' msg - too Slow
00000005          00076 Time_Bit        equ  5    ;Indicate 512 uS has passed
                  00077
                  00078 ;Equates for LED Task #7 - LED_Mode
00000003          00079 LED_B           equ  3              ;LED is active - LED_Mode.3
00000007          00080 LED_NEW_B       equ  7              ;LED has just changed Modes = 1
00000008          00081 LED_OFF_MODE    equ  B'00001000'    ;LED OFF
00000089          00082 LED_SEQ1_MODE   equ  B'10001001'    ;LED Sequence 1: .2s On, 1s Off
0000008A          00083 LED_SEQ2_MODE   equ  B'10001010'    ;LED Sequence 2: 3x(.2s), 1s Off
0000008B          00084 LED_SEQ3_MODE   equ  B'10001011'    ;LED Sequence 3: 5x(.2s), 1s Off
0000009C          00085 LED_SLOW_MODE   equ  B'10011100'    ;LED Slow Pulsing - .3 Hz
0000009D          00086 LED_MEDIUM_MODE equ  B'10011101'    ;LED Medium Pulsing - 1 Hz
0000009E          00087 LED_FAST_MODE   equ  B'10011110'    ;LED Fast Pulsing - 3 Hz
0000008F          00088 LED_ON_MODE     equ  B'10001111'    ;LED ON Continuously
                  00089
0000              00090         ORG     0                   ;Reset Vector
                  00091
0000 28F7         00092         GOTO    Main
                  00093
0004              00094         ORG     4                   ;Interrupt Vector
                  00095
0004 29D4         00096         GOTO    Interrupt
                  00097
                  00098 ;******          Task #1 - Asynchronous 9600 Baud Serial Output (LOW=0)
0005              00099 Do_OState
0005 0828         00100         MOVF    OState,W            ;Get (0-A) mode #
0006 390F         00101         ANDLW   H'0F'               ;Get only mode #
0007 0782         00102         ADDWF   PCL, F              ;jump to subroutine
0008 2814         00103         GOTO    OStateS             ;Serial Start Bit
0009 2823         00104         GOTO    OState0_7           ;Bit 0
000A 2823         00105         GOTO    OState0_7           ;Bit 1
000B 2823         00106         GOTO    OState0_7           ;Bit 2
000C 2823         00107         GOTO    OState0_7           ;Bit 3
000D 2823         00108         GOTO    OState0_7           ;Bit 4
000E 2823         00109         GOTO    OState0_7           ;Bit 5
000F 2823         00110         GOTO    OState0_7           ;Bit 6
0010 2823         00111         GOTO    OState0_7           ;Bit 7
0011 2829         00112         GOTO    OStateE             ;Serial Stop Bit
0012 2831         00113         GOTO    OStateL             ;Last State
0013 0008         00114         RETURN
                  00115
0014              00116 OStateS
0014 3000         00117         MOVLW   H'00'               ;Post & Pre 1=1 & OFF
```


© 1997 Microchip Technology Inc.

```
0015 0092           00118       MOVWF   T2CON
0016 1683           00119       BSF     STATUS,RP0      ;Point to BANK 1
0017 3053           00120       MOVLW   H'68' - H'15'   ;104uS - 9600 Baud & adjust to Latency
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0018 0092           00121       MOVWF   PR2
0019 1283           00122       BCF     STATUS,RP0      ;Point to BANK 0
001A 0191           00123       CLRF    TMR2            ;Init to 0
001B 108C           00124       BCF     PIR1,TMR2IF     ;Clear Timer 2 Flag so as to start fresh
001C 1505           00125       BSF     PORTA,Serial_Out;Output 'Serial Start Bit' starting Now
001D 1512           00126       BSF     T2CON,TMR2ON    ;Start Timer 2
001E 0AA8           00127       INCF    OState, F  ;inc to next state BEFORE allowing interrupt
                    00128
001F 1683           00129       BSF     STATUS,RP0      ;Point to BANK 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0020 148C           00130       BSF     PIE1,TMR2IE     ;Allow for Timer 2 interrupts
0021 1283           00131       BCF     STATUS,RP0      ;Point to BANK 0
0022 0008           00132       RETURN
                    00133
0023                00134 OState0_7                     ;Bit 0 - 7
0023 0CB1           00135       RRF     xmt_byte, F     ;Move bit into C from right most bit
0024 1C03           00136       BTFSS   STATUS,C        ;Output C bit
0025 1105           00137       BCF     PORTA,Serial_Out  ;
0026 1803           00138       BTFSC   STATUS,C          ;
0027 1505           00139       BSF     PORTA,Serial_Out  ;
0028 282A           00140       GOTO    OS_End
0029                00141 OStateE
0029 1105           00142         BCF     PORTA,Serial_Out  ;Serial Stop Bit
002A 108C           00143 OS_End  BCF     PIR1,TMR2IF     ;Clear Timer 2 Flag so as to start fresh
002B 1683           00144         BSF     STATUS,RP0      ;Point to BANK 1
002C 306C           00145         MOVLW   H'68' + H'4     ;104uS - 9600 Baud & adjust to Latency
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
002D 0092           00146         MOVWF   PR2
002E 1283           00147         BCF     STATUS,RP0      ;Point to BANK 0
002F 0AA8           00148         INCF    OState, F       ;increment to next state
0030 0008           00149         RETURN
0031                00150 OStateL
0031 1683           00151         BSF     STATUS,RP0      ;Point to BANK 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0032 108C           00152         BCF     PIE1,TMR2IE     ;Do NOT Allow Timer 2 interrupts
0033 1283           00153         BCF     STATUS,RP0      ;Point to BANK 0
                    00154
0034 108C           00155         BCF     PIR1,TMR2IF     ;Clear Timer 2 Flag so as to start fresh
0035 1112           00156         BCF     T2CON,TMR2ON    ;Stop Timer 2
0036 01A8           00157         CLRF    OState          ;Ready to send next byte out
0037 1025           00158         BCF     T_B,OState_B    ;Serial Out not active
0038 0008           00159         RETURN
                    00160
                    00161 ;******  Task #7 - Output Highest Level Indication on LED
0039                00162 Do_LED
0039 1BA7           00163         BTFSC   LED_Mode,LED_NEW_B;Initialize regs if change in modes
003A 284C           00164         GOTO    LED_NEW
003B 0AB0           00165         INCF    LED_C, F        ;Inc Counter - Time Unit = 131072 uS
003C 0827           00166         MOVF    LED_Mode,W      ;Get (0-7) mode #
003D 3907           00167         ANDLW   H'07'           ;Get only mode #
003E 0782           00168         ADDWF   PCL, F          ;jump to subroutine
003F 2848           00169         GOTO    LED_OFF         ;LED OFF
0040 2864           00170         GOTO    LED_SEQ1        ;LED Seq 1: 1 short pulse & pause
0041 2867           00171         GOTO    LED_SEQ2        ;LED Seq 2: 2 short pulses & pause
0042 288A           00172         GOTO    LED_SEQ3        ;LED Seq 3: 3 short pulses & pause
0043 2850           00173         GOTO    LED_SLOW        ;LED Slow Pulsing - .3 Hz
0044 285E           00174         GOTO    LED_MEDIUM      ;LED Medium Pulsing - 1 Hz
0045 2861           00175         GOTO    LED_FAST        ;LED Fast Pulsing - 3 Hz
0046 284D           00176         GOTO    LED_ON          ;LED ON Continuously
0047 0008           00177 _0012   RETURN
                    00178 ;------
0048                00179 LED_OFF
```

5

```
0048 1085          00180          BCF     PORTA,LED        ;Turn off LED
0049 11A7          00181          BCF     LED_Mode,LED_B   ;LED must be off
004A 01B0          00182          CLRF    LED_C            ;Reset Counter - LED_C = 0
004B 0008          00183          RETURN
                   00184 ;------
004C               00185 LED_NEW
004C 13A7          00186          BCF     LED_Mode,LED_NEW_B  ;Done initializing
004D               00187 LED_ON
004D 1485          00188          BSF     PORTA,LED        ;Turn on LED
004E 01B0          00189          CLRF    LED_C            ;Reset Counter - LED_C = 0
004F 0008          00190          RETURN
                   00191 ;------
0050               00192 LED_SLOW
0050 300C          00193          MOVLW   H'0C'            ;.3Hz @ 50% Duty
0051 00A0          00194          MOVWF   temp
0052 0820          00195 LED_S    MOVF    temp,W           ;Check LED_C if time, .3Hz @ 50% Duty
0053 0230          00196          SUBWF   LED_C,W
0054 1D03          00197          BTFSS   STATUS,Z
0055 2847          00198          GOTO    _0012
0056 3010          00199          MOVLW   H'10'
0057 06A7          00200          XORWF   LED_Mode, F      ;Switch states
0058 1E27          00201          BTFSS   LED_Mode,4       ;Now make LED same state
0059 1085          00202          BCF     PORTA,LED
005A 1A27          00203          BTFSC   LED_Mode,4
005B 1485          00204          BSF     PORTA,LED
005C 01B0          00205          CLRF    LED_C            ;Reset LED_C
005D 0008          00206          RETURN
                   00207 ;------
005E               00208 LED_MEDIUM
005E 3004          00209          MOVLW   H'04'            ;1Hz @ 50% Duty
005F 00A0          00210          MOVWF   temp
0060 2852          00211          GOTO    LED_S            ;Go do it
                   00212 ;------
0061               00213 LED_FAST
0061 3001          00214          MOVLW   H'01'            ;3Hz @ 50% Duty
0062 00A0          00215          MOVWF   temp
0063 2852          00216          GOTO    LED_S            ;Go do it
                   00217 ;------
0064               00218 LED_SEQ1                          ;.2 ON, 1 OFF
0064 1E27          00219          BTFSS   LED_Mode,4       ;Skip if bit is high
0065 2876          00220          GOTO    ON1              ;Go do it
0066 2882          00221          GOTO    OFF3             ;Go do it
                   00222 ;------
0067               00223 LED_SEQ2                          ;.2 ON, .2 OFF, .2 ON, 1 OFF
0067 0827          00224          MOVF    LED_Mode,W
0068 00A0          00225          MOVWF   temp
0069 3030          00226          MOVLW   H'30'            ;Get sequence # only
006A 05A0          00227          ANDWF   temp, F
006B 0EA0          00228          SWAPF   temp, F          ;swap nibbles
006C 0820          00229          MOVF    temp,W           ;get nibble for offset
006D 0782          00230          ADDWF   PCL, F           ;Table jump calculation
006E 2876          00231          GOTO    ON1              ;LED is on, check if time to change
006F 287C          00232          GOTO    OFF2             ;LED is off, check if time to change
0070 2876          00233          GOTO    ON1              ;LED is on, check if time to change
0071 2882          00234          GOTO    OFF3             ;LED is off, check if time to change
                   00235 ;------
0072               00236 LED_Exit
0072 3010          00237          MOVLW   H'10'            ;Inc Seq #
0073 07A7          00238          ADDWF   LED_Mode, F
0074 01B0          00239          CLRF    LED_C            ;Reset LED_C
0075 0008          00240          RETURN
0076               00241 ON1
0076 3002          00242          MOVLW   H'02'            ;Check LED_C if time, .2 sec-on
0077 0230          00243          SUBWF   LED_C,W
0078 1D03          00244          BTFSS   STATUS,Z
0079 2847          00245          GOTO    _0012
```


© 1997 Microchip Technology Inc.

```
007A 1085          00246          BCF     PORTA,LED       ;Turn off LED
007B 2872          00247          GOTO    LED_Exit
007C               00248 OFF2
007C 3002          00249          MOVLW   H'02'           ;Check LED_C if time, .2 sec-on
007D 0230          00250          SUBWF   LED_C,W
007E 1D03          00251          BTFSS   STATUS,Z
007F 2847          00252          GOTO    _0012
0080 1485          00253          BSF     PORTA,LED       ;Turn on LED
0081 2872          00254          GOTO    LED_Exit
0082               00255 OFF3
0082 3008          00256          MOVLW   H'08'           ;Check LED_C if time, 1 sec-off
0083 0230          00257          SUBWF   LED_C,W
0084 1D03          00258          BTFSS   STATUS,Z
0085 2847          00259          GOTO    _0012
0086 1485          00260          BSF     PORTA,LED       ;Turn on LED
0087 30F0          00261          MOVLW   H'F0'
0088 04A7          00262          IORWF   LED_Mode, F     ;Cause (Seq# & NEW) to overflow to 0
0089 2872          00263          GOTO    LED_Exit
008A               00264 LED_SEQ3      ;.2 ON, .2 OFF, .2 ON, .2 OFF, .2 ON, 1 OFF
008A 0827          00265          MOVF    LED_Mode,W      ;Get LED info
008B 00A0          00266          MOVWF   temp
008C 3070          00267          MOVLW   H'70'           ;Get sequence # only
008D 05A0          00268          ANDWF   temp, F
008E 0EA0          00269          SWAPF   temp, F         ;swap nibbles
008F 0820          00270          MOVF    temp,W          ;get nibble for offset
0090 0782          00271          ADDWF   PCL, F          ;Table jump calculation
0091 2876          00272          GOTO    ON1             ;LED is on check if time to change
0092 287C          00273          GOTO    OFF2            ;LED is off check if time to change
0093 2876          00274          GOTO    ON1             ;LED is on check if time to change
0094 287C          00275          GOTO    OFF2            ;LED is off check if time to change
0095 2876          00276          GOTO    ON1             ;LED is on check if time to change
0096 2882          00277          GOTO    OFF3            ;LED is off check if time to change
                   00278
                   00279 ;******          Task #2 - Asynchronous 4800 Baud Serial Input (LOW=0)
0097               00280 Do_I1State
0097 0829          00281          MOVF    IState1,W       ;Get (0-B) mode #
0098 390F          00282          ANDLW   H'0F'           ;Get only mode #
0099 0782          00283          ADDWF   PCL, F          ;jump to subroutine
009A 28A7          00284          GOTO    I1StateS        ;Serial Start Bit
009B 28B8          00285          GOTO    I1State2        ;1/2 of Start Bit - see if False Start
009C 28C5          00286          GOTO    I1State0_7      ;Bit 0
009D 28C5          00287          GOTO    I1State0_7      ;Bit 1
009E 28C5          00288          GOTO    I1State0_7      ;Bit 2
009F 28C5          00289          GOTO    I1State0_7      ;Bit 3
00A0 28C5          00290          GOTO    I1State0_7      ;Bit 4
00A1 28C5          00291          GOTO    I1State0_7      ;Bit 5
00A2 28C5          00292          GOTO    I1State0_7      ;Bit 6
00A3 28C5          00293          GOTO    I1State0_7      ;Bit 7
00A4 28D4          00294          GOTO    I1StateE        ;Serial Stop Bit
00A5 28DF          00295          GOTO    I1StateL        ;Last State - End of Stop Bit
00A6 0008          00296          RETURN
                   00297
                   00298 ;***             Subroutines for Task #2
00A7               00299 I1StateS                         ;Start Bit - Setup timing variables
00A7 120B          00300       BCF     INTCON,INTE        ;Disable detecting changes on INT pin
00A8 108B          00301       BCF     INTCON,INTF        ;Clear Interrupting Flag
00A9 3000          00302       MOVLW   H'00'              ;Internal Clk, Pre 1=1 & OFF
00AA 0090          00303       MOVWF   T1CON
00AB 018E          00304       CLRF    TMR1L      ;Calculate (0 - #) of counts until roll-over
00AC 304A          00305       MOVLW   H'68' - H'1E' ;208us/2 = 4800 Baud & adjust to Latency
00AD 028E          00306       SUBWF   TMR1L, F
00AE 018F          00307       CLRF    TMR1H
00AF 038F          00308       DECF    TMR1H, F           ;H'FF'
00B0 100C          00309       BCF     PIR1,TMR1IF        ;Clear Timer 1 Flag so as to start fresh
00B1 1410          00310       BSF     T1CON,TMR1ON       ;Start Timer 1
00B2 0AA9          00311       INCF    IState1, F ;inc to next state BEFORE allowing interrupts
```

```
00B3 14A5          00312       BSF     T_B,IState1_B   ;Serial Input Active
                   00313
00B4 1683          00314       BSF     STATUS,RP0      ;Point to BANK 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
00B5 140C          00315       BSF     PIE1,TMR1IE     ;Allow for Timer 1 interrupts
00B6 1283          00316       BCF     STATUS,RP0      ;Point to BANK 0
00B7 0008          00317       RETURN
00B8               00318 I1State2                      ;Check if still a Start Bit
00B8 1C06          00319       BTFSS   PORTB,Serial_IN_1  ;False Start Error ?
00B9 28E9          00320       GOTO    FS_Error_1
00BA 1026          00321       BCF     FLAGS,FS_Flag_1 ;Start Bit OK
00BB 1010          00322       BCF     T1CON,TMR1ON    ;Stop Timer 1
00BC 018E          00323       CLRF    TMR1L
00BD 30AA          00324       MOVLW   H'D0' - H'26'   ;208us = 4800 Baud & adjust to Latency
00BE 028E          00325       SUBWF   TMR1L, F
00BF 018F          00326       CLRF    TMR1H
00C0 038F          00327       DECF    TMR1H, F        ;H'FF'
00C1 100C          00328       BCF     PIR1,TMR1IF     ;Clear Timer 1 Flag so as to start fresh
00C2 1410          00329       BSF     T1CON,TMR1ON    ;Start Timer 1
00C3 0AA9          00330       INCF    IState1, F      ;increment to next state
00C4 0008          00331       RETURN
00C5               00332 I1State0_7                    ;Bit 0 - 7
00C5 1C06          00333       BTFSS   PORTB,Serial_IN_1  ;Move Input bit into C
00C6 1003          00334       BCF     STATUS,C
00C7 1806          00335       BTFSC   PORTB,Serial_IN_1
00C8 1403          00336       BSF     STATUS,C
00C9 0CB2          00337       RRF     rcv_byte_1, F   ;Move C into left most bit
00CA 1010          00338       BCF     T1CON,TMR1ON    ;Stop Timer 1
00CB 018E          00339       CLRF    TMR1L
00CC 30AA          00340       MOVLW   H'D0' - H'26'   ;208us = 4800 Baud & adjust to Latency
00CD 028E          00341       SUBWF   TMR1L, F
00CE 018F          00342       CLRF    TMR1H
00CF 038F          00343       DECF    TMR1H, F        ;H'FF'
00D0 100C          00344       BCF     PIR1,TMR1IF     ;Clear Timer 1 Flag so as to start fresh
00D1 1410          00345       BSF     T1CON,TMR1ON    ;Start Timer 1
00D2 0AA9          00346       INCF    IState1, F      ;increment to next state
00D3 0008          00347       RETURN
00D4               00348 I1StateE                      ;Check if we have a proper Stop Bit
00D4 1806          00349       BTFSC   PORTB,Serial_IN_1  ;Frame Error
00D5 28EC          00350       GOTO    F_Error_1
00D6 10A6          00351       BCF     FLAGS,FE_Flag_1 ;Stop Bit OK
00D7 01AD          00352       CLRF    T_5_S_CO        ;Reset 5 Sec Timer - got a good byte
                   00353       ;Process the msg Here !
00D8 0832          00354       MOVF    rcv_byte_1,W    ;Make a copy of just received byte
00D9 00B3          00355       MOVWF   RCV_Storage
00DA 1E25          00356       BTFSS   T_B,RCV_Got_One_B  ;Report Lost data
00DB 1126          00357       BCF     FLAGS,RCV_Overflow
00DC 1A25          00358       BTFSC   T_B,RCV_Got_One_B
00DD 1526          00359       BSF     FLAGS,RCV_Overflow
00DE 1625          00360       BSF     T_B,RCV_Got_One_B ;We Now have a RB Value to go out
00DF               00361 I1StateL
00DF 1010          00362       BCF     T1CON,TMR1ON    ;Stop Timer 1
00E0 1683          00363       BSF     STATUS,RP0      ;Point to BANK 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
00E1 140C          00364       BSF     PIE1,TMR1IE     ;Allow for Timer 1 interrupts
00E2 1283          00365       BCF     STATUS,RP0      ;Point to BANK 0
00E3 100C          00366       BCF     PIR1,TMR1IF     ;Clear Timer 1 Flag so as to start fresh
                   00367
00E4 01A9          00368       CLRF    IState1         ;Ready to receive next byte
00E5 10A5          00369       BCF     T_B,IState1_B   ;Serial In not currently active
                   00370
00E6 108B          00371       BCF     INTCON,INTF     ;Clear Interrupting Flag
00E7 160B          00372       BSF     INTCON,INTE     ;Enable detecting changes on INT pin
00E8 0008          00373       RETURN
00E9               00374 FS_Error_1    ;False Start - Shut Down Checking
00E9 10A5          00375       BCF     T_B,IState1_B   ;Serial Input NOT Active
```


© 1997 Microchip Technology Inc.

```
00EA 1426           00376       BSF     FLAGS,FS_Flag_1     ;False Start Error
00EB 28DF           00377       GOTO    I1StateL            ;Start All Over
00EC                00378 F_Error_1      ;Frame Error - Wait for End of Stop Bit
00EC 14A6           00379       BSF     FLAGS,FE_Flag_1     ;Frame Error for this Byte ?
00ED 0AA9           00380       INCF    IState1, F          ;Increment to next state
00EE 1010           00381       BCF     T1CON,TMR1ON        ;Stop Timer 1
00EF 018E           00382       CLRF    TMR1L
00F0 304A           00383       MOVLW   H'68' - H'1E'       ;208us/2 = 4800 Baud & adjust to Latency
00F1 028E           00384       SUBWF   TMR1L, F
00F2 018F           00385       CLRF    TMR1H
00F3 038F           00386       DECF    TMR1H, F            ;H'FF'
00F4 100C           00387       BCF     PIR1,TMR1IF         ;Clear Timer 1 Flag so as to start fresh
00F5 1410           00388       BSF     T1CON,TMR1ON        ;Start Timer 1
00F6 0008           00389       RETURN
                    00390
                    00391 ;******          Code Starting point
00F7                00392 Main
00F7 0183           00393       CLRF    STATUS
00F8 0184           00394       CLRF    FSR
00F9 0181           00395       CLRF    TMR0                ;Clear Timer0
00FA 3000           00396       MOVLW   H'00'               ;What is High/Low for RA at RESET State
00FB 0085           00397       MOVWF   PORTA
00FC 0086           00398       MOVWF   PORTB
00FD 0087           00399       MOVWF   PORTC
00FE 0088           00400       MOVWF   PORTD
00FF 0089           00401       MOVWF   PORTE
0100 018A           00402       CLRF    PCLATH
0101 3060           00403       MOVLW   H'60'       ;/GIE,PEIE,T0IE,/INTE,/RBIE,/T0IF,/INTF,/RBIF
0102 008B           00404       MOVWF   INTCON
0103 018C           00405       CLRF    PIR1            ;Timer 2 Flag cleared
0104 018E           00406       CLRF    TMR1L
0105 018F           00407       CLRF    TMR1H
0106 0190           00408       CLRF    T1CON           ;Timer 1 OFF until ready for input
0107 0191           00409       CLRF    TMR2
0108 0192           00410       CLRF    T2CON           ;Timer 2 OFF until have byte to output
0109 0193           00411       CLRF    SSPBUF
010A 0194           00412       CLRF    SSPCON
010B 0195           00413       CLRF    CCPR1L
010C 0196           00414       CLRF    CCPR1H
010D 0197           00415       CLRF    CCP1CON
                    00416
010E 1683           00417       BSF     STATUS,RP0      ;Point to BANK 1
010F 3040           00418       MOVLW   H'40'   ;TMR0/2 & Interrupt on Rising edge of INT
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0110 0081           00419       MOVWF   OPTION_REG      ;Load OPTION reg
0111 30F1           00420       MOVLW   RA_TRIS
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0112 0085           00421       MOVWF   TRISA
0113 30FF           00422       MOVLW   H'FF'           ;RB TRIS at RESET State
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0114 0086           00423       MOVWF   TRISB
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0115 0087           00424       MOVWF   TRISC
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0116 0088           00425       MOVWF   TRISD
0117 3007           00426       MOVLW   H'07'           ;PSPMODE=0
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0118 0089           00427       MOVWF   TRISE
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0119 018C           00428       CLRF    PIE1            ;Timer 2 Interrupt disabled
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
011A 148E           00429       BSF     PCON,NOT_POR
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
011B 0192           00430       CLRF    PR2
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
011C 0193           00431       CLRF    SSPADD
```


© 1997 Microchip Technology Inc.

```
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
011D 0194           00432       CLRF    SSPSTAT
                    00433
011E 1283           00434       BCF     STATUS,RP0  ;Point to BANK 0
011F 21BF           00435       CALL    Clear_Regs  ;Clear Registers 20-7F, A0-C0 Memory Pages
                    00436
                    00437 ;Initialize Tasks
                    00438 ;Task #1 waits for byte to output
                    00439 ;Task #2 waits for Serial IN Start Bit
0120 3031           00440       MOVLW   H'31'       ;Task #3 is initialized for square pulses
0121 00B6           00441       MOVWF   PWM_ON          ;   "   25 mS ON
0122 3031           00442       MOVLW   H'31'           ;   "   Period = 50 mS, DS= 50%
0123 00B7           00443       MOVWF   PWM_OFF         ;   "  25 mS OFF
0124 01AF           00444       CLRF    T_PWM_CO        ;   "
0125 0836           00445       MOVF    PWM_ON,W        ;move PWM_tmp,PWM_ON
0126 00B8           00446       MOVWF   PWM_tmp         ; "
0127 1585           00447       BSF     PORTA,PWM_Out   ;Start Outputting ON
0128 17A5           00448       BSF     T_B,T_PWM_B     ;   "
                    00449                               ;Task #4 runs when Task 1 is Not
                    00450                               ;Task #5 is always running
0129 0806           00451       MOVF    PORTB,W         ;Task #6 is Initialized here
012A 00B4           00452       MOVWF   Old_RB
012B 0834           00453       MOVF    Old_RB,W        ;Make all the same initial value
012C 00B5           00454       MOVWF   Last_RB
012D 16A5           00455       BSF     T_B,RB_NEW_B    ;Tell Task #4: RB byte ready to output
012E 3008           00456       MOVLW   LED_OFF_MODE
012F 00A7           00457       MOVWF   LED_Mode        ;Task #7 is Started
0130 1525           00458       BSF     T_B,T_5_S_B     ;Task #8 is Started here
0131 15A5           00459       BSF     T_B,T_5_M_B     ;Task #9 is Started here
                    00460
0132 178B           00461       BSF     INTCON,GIE      ;Enable Global Interrupts
                    00462
                    00463 ; Handle Task & Timer activities - Main Loop done in background
0133                00464 Inc_Time      ;Heart Beat - Time unit = 512 uS for Tasks #5 & #6
0133 1EA6           00465       BTFSS   FLAGS,Time_Bit ;Idle Task - wait until 512 uS has gone by
0134 2933           00466       GOTO    Inc_Time
0135 12A6           00467       BCF     FLAGS,Time_Bit ;Reset for next indicator
                    00468                              ;from TMR0 Interrupt
0136                00469 Task_3       ;Task #3 - PWM, Period = (PWM_ON + PWM_OFF) * 512uS
0136 1FA5           00470       BTFSS   T_B,T_PWM_B     ;if NOT outputting now then skip call
0137 2947           00471       GOTO    Task_4
0138 0AAF           00472       INCF    T_PWM_CO, F     ;Inc count of time
0139 0838           00473       MOVF    PWM_tmp,W       ;cjne T_PWM_CO,PWM_tmp,Task_4
013A 022F           00474       SUBWF   T_PWM_CO,W      ; "
013B 1D03           00475       BTFSS   STATUS,Z        ; "
013C 2947           00476       GOTO    Task_4          ; "
013D 01AF           00477       CLRF    T_PWM_CO        ;Reset timer
013E 1985           00478       BTFSC   PORTA,PWM_Out
013F 2944           00479       GOTO    T3_1
0140 1585           00480       BSF     PORTA,PWM_Out   ;Change Output State
0141 0836           00481       MOVF    PWM_ON,W        ;move PWM_tmp,PWM_ON
0142 00B8           00482       MOVWF   PWM_tmp         ; "
0143 2947           00483       GOTO    Task_4
0144                00484 T3_1
0144 1185           00485       BCF     PORTA,PWM_Out   ;Change Output State
0145 0837           00486       MOVF    PWM_OFF,W       ;mov PWM_tmp,PWM_OFF
0146 00B8           00487       MOVWF   PWM_tmp         ; "
                    00488
0147                00489 Task_4  ;Task #4 - Finds next Buffered Byte to Send Out through Task 1
0147 1825           00490       BTFSC   T_B,OState_B    ;if outputting now then skip call
0148 295E           00491       GOTO    Task_5
0149 1E25           00492       BTFSS   T_B,RCV_Got_One_B  ;Got a NEW Received byte to send
014A 2950           00493       GOTO    _0057
014B 0833           00494       MOVF    RCV_Storage,W      ;Send just received byte
014C 00B1           00495       MOVWF   xmt_byte
014D 1225           00496       BCF     T_B,RCV_Got_One_B  ;Clear need to send old byte
```


© 1997 Microchip Technology Inc.

```
014E 1425           00497        BSF     T_B,OState_B       ;Start Task #1 & Lock Out Others
014F 295D           00498        GOTO    T4_S
0150 1EA5           00499 _0057  BTFSS   T_B,RB_NEW_B       ;Indicates a change in RB input
0151 2957           00500        GOTO    _0058
0152 0834           00501        MOVF    Old_RB,W           ;Send New RB value
0153 00B1           00502        MOVWF   xmt_byte
0154 12A5           00503        BCF     T_B,RB_NEW_B       ;Clear need to send out newest value
0155 1425           00504        BSF     T_B,OState_B       ;Start Task #1 & Lock Out Others
0156 295D           00505        GOTO    T4_S
0157 1F25           00506 _0058  BTFSS   T_B,S_5_S_B        ;Serial In 5 secs of inactivity
0158 295E           00507        GOTO    Task_5
0159 30FF           00508        MOVLW   H'FF'              ;Tell of inactivity of Serial In
015A 00B1           00509        MOVWF   xmt_byte
015B 1325           00510        BCF     T_B,S_5_S_B        ;Clear need to send msg
015C 1425           00511        BSF     T_B,OState_B       ;Start Task #1 & Lock Out Others
015D                00512 T4_S                              ;Start Task #1
015D 2005           00513          CALL    Do_OState
                    00514
015E                00515 Task_5   ;Task #5 - Monitor Level Reset Input Line - Always Running !
015E 1805           00516          BTFSC   PORTA,Level_Reset
015F 2962           00517          GOTO    Task_6
0160 3008           00518          MOVLW   LED_OFF_MODE       ;Lowest Level Indicator output
0161 00A7           00519          MOVWF   LED_Mode
                    00520
0162                00521 Task_6 ;Task #6 - Debounce 8 bit Input Sensors - Runs every 20 mS
0162 0AAE           00522          INCF    T_20_mS_CO, F      ;Inc Counter - Time Unit = 512 uS
0163 3027           00523          MOVLW   H'27'              ;Used to debounce the input
0164 022E           00524          SUBWF   T_20_mS_CO,W
0165 1D03           00525          BTFSS   STATUS,Z
0166 297C           00526          GOTO    _0065
0167 01AE           00527          CLRF    T_20_mS_CO         ;Reset T_20_mS_CO to start over again
0168 0806           00528          MOVF    PORTB,W            ;Last copy of RB same as Current ?
0169 0235           00529          SUBWF   Last_RB,W
016A 1903           00530          BTFSC   STATUS,Z
016B 296F           00531          GOTO    _0062
016C 0806           00532          MOVF    PORTB,W            ;Store Current RB - diff from Last
016D 00B5           00533          MOVWF   Last_RB
016E 2971           00534          GOTO    _0063
016F 0835           00535 _0062    MOVF    Last_RB,W          ;New Old RB <- same value over 20 mS
0170 00B4           00536          MOVWF   Old_RB
0171 08B4           00537 _0063    MOVF    Old_RB, F          ;See if RB is now 0
0172 1903           00538          BTFSC   STATUS,Z           ;RB == 0 ? then keep timer running
0173 2976           00539          GOTO    _0064
0174 01AB           00540          CLRF    T_5_M_LO           ;Reset 5 Min Timer
0175 01AC           00541          CLRF    T_5_M_HI           ;  still not zero yet
0176 21A8           00542 _0064    CALL    D_H_E_L            ;Determine the Highest Error Level
0177 1EA5           00543          BTFSS   T_B,RB_NEW_B       ;Check for Lost Data Error
0178 11A6           00544          BCF     FLAGS,RB_Overflow
0179 1AA5           00545          BTFSC   T_B,RB_NEW_B
017A 15A6           00546          BSF     FLAGS,RB_Overflow
017B 16A5           00547          BSF     T_B,RB_NEW_B       ;Every 20 mS send Old_RB out
                    00548
                    00549        ;Heart Beat - Time unit = 131072 uS for Tasks #7, #8 & #9
017C                00550 _0065
017C 1683           00551          BSF     STATUS,RP0         ;Point to BANK 1
017D 30F1           00552          MOVLW   RA_TRIS            ;RA TRIS - refresh
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
017E 0085           00553          MOVWF   TRISA
017F 30FF           00554          MOVLW   H'FF'              ;RB TRIS - refresh
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0180 0086           00555          MOVWF   TRISB
0181 1283           00556          BCF     STATUS,RP0         ;Point to BANK 0
0182 0BAA           00557          DECFSZ  cc, F              ;Step-up time units * 256
0183 2933           00558          GOTO    Inc_Time
                    00559
0184                00560 Task_7   ;Task 7 - Output Highest Level Indication on LED
```


© 1997 Microchip Technology Inc.

```
0184 1DA7          00561         BTFSS   LED_Mode,LED_B   ;Is LED active ?
0185 2987          00562         GOTO    Task_8
0186 2039          00563         CALL    Do_LED           ;Handle LED timing
                   00564
0187               00565 Task_8 ;Task #8 - 5 Second Serial Input Lack of Activity Timer
0187 1D25          00566         BTFSS   T_B,T_5_S_B      ;5 Sec Timer Active ?
0188 2996          00567         GOTO    Task_9
0189 0AAD          00568         INCF    T_5_S_CO, F      ;Inc Counter - Time Unit = 131072 uS
018A 3026          00569         MOVLW   H'26'            ;Check T_5_S_CO if time
018B 022D          00570         SUBWF   T_5_S_CO,W
018C 1D03          00571         BTFSS   STATUS,Z
018D 2996          00572         GOTO    Task_9
018E 01AD          00573         CLRF    T_5_S_CO         ;Reset T_5_S_CO
018F 308F          00574         MOVLW   LED_ON_MODE      ;Highest Level Indicator output
0190 00A7          00575         MOVWF   LED_Mode
0191 1F25          00576         BTFSS   T_B,S_5_S_B      ;Check if Lost Data Error
0192 1226          00577         BCF     FLAGS,S_5_S_Overflow
0193 1B25          00578         BTFSC   T_B,S_5_S_B
0194 1626          00579         BSF     FLAGS,S_5_S_Overflow
0195 1725          00580         BSF     T_B,S_5_S_B      ;Send notice of 5 seconds of inaction
                   00581
0196               00582 Task_9 ;Task #9 - 5 Min. Lack of Severe Error from Sensors Reset Timer
0196 1DA5          00583         BTFSS   T_B,T_5_M_B      ;5 Min Timer Active ?
0197 29A7          00584         GOTO    Task_A
0198 0AAB          00585         INCF    T_5_M_LO, F      ;Inc LO Counter; Time Unit = 131072 uS
0199 1903          00586         BTFSC   STATUS,Z         ;See if carry needs to be passed on ?
019A 0AAC          00587         INCF    T_5_M_HI, F      ;Inc HI Counter; Time Unit = 131072 uS
019B 3008          00588         MOVLW   H'08'            ;#2288<  Check T_5_M_HI if time
019C 022C          00589         SUBWF   T_5_M_HI,W
019D 1D03          00590         BTFSS   STATUS,Z
019E 29A7          00591         GOTO    Task_A
019F 30F0          00592         MOVLW   H'F0'            ;#2288>  Check T_5_M_LO if time
01A0 022B          00593         SUBWF   T_5_M_LO,W
01A1 1D03          00594         BTFSS   STATUS,Z
01A2 29A7          00595         GOTO    Task_A
01A3 01AB          00596         CLRF    T_5_M_LO         ;Reset T_5_M_LO
01A4 01AC          00597         CLRF    T_5_M_HI         ;Reset T_5_M_HI
01A5 3008          00598         MOVLW   LED_OFF_MODE     ;Lowest Level Indicator output
01A6 00A7          00599         MOVWF   LED_Mode
01A7               00600 Task_A
01A7 2933          00601         GOTO    Inc_Time         ;Loop Forever
                   00602
                   00603 ;****           Determine the Highest Error Level & Start Task #7
01A8               00604 D_H_E_L
01A8 3007          00605         MOVLW   H'07'            ;Check top 7 bits
01A9 00A0          00606         MOVWF   temp
01AA 0834          00607         MOVF    Old_RB,W         ;Get copy of 7 debounced Sensor Input
01AB 00A1          00608         MOVWF   tmp
01AC 0DA1          00609 _0070   RLF     tmp, F           ;Put top bit into C bit
01AD 1803          00610         BTFSC   STATUS,C         ;Check if C bit is set
01AE 29B4          00611         GOTO    _0072
01AF 0BA0          00612         DECFSZ  temp, F          ;Continue to check lesser bits
01B0 29AC          00613         GOTO    _0070
01B1 0806          00614 _0071   MOVF    PORTB,W          ;Restore current value of RB
01B2 00A1          00615         MOVWF   tmp
01B3 0008          00616         RETURN
01B4 0827          00617 _0072   MOVF    LED_Mode,W       ;Get current Level Indicator
01B5 3907          00618         ANDLW   H'07'            ;Get only      "       "
01B6 00A1          00619         MOVWF   tmp              ;Store into a temporary register
01B7 0820          00620         MOVF    temp,W           ;Check if already at this Level
01B8 0221          00621         SUBWF   tmp,W
01B9 1803          00622         BTFSC   STATUS,C
01BA 29B1          00623         GOTO    _0071
01BB 3088          00624         MOVLW   H'88'            ;Start to build LED_Mode
01BC 0420          00625         IORWF   temp,W           ;Put new Level Indicator into reg
01BD 00A7          00626         MOVWF   LED_Mode         ;Store new LED Mode
```


© 1997 Microchip Technology Inc.

```
01BE 29B1          00627          GOTO    _0071
                   00628
                   00629 ;******          Clear Registers 20-7Fh, A0-C0
01BF               00630 Clear_Regs
01BF 307F          00631          MOVLW   H'7F'          ;First regs to clear in Bank 0
01C0 0084          00632          MOVWF   FSR
01C1 0180          00633 Loop_C1  CLRF    INDF           ;Clear reg
01C2 0384          00634          DECF    FSR, F         ;point to next reg to clear
01C3 3020          00635          MOVLW   H'20'          ;Dec temp, jump if not done
01C4 0204          00636          SUBWF   FSR,W
01C5 1803          00637          BTFSC   STATUS,C
01C6 29C1          00638          GOTO    Loop_C1
                   00639
01C7 30C0          00640          MOVLW   H'C0'          ;First regs to clear in Bank 1
01C8 0084          00641          MOVWF   FSR
01C9 0180          00642 Loop_C2  CLRF    INDF           ;Clear reg
01CA 0384          00643          DECF    FSR, F         ;point to next reg to clear
01CB 30A0          00644          MOVLW   H'A0'          ;Dec temp, jump if not done
01CC 0204          00645          SUBWF   FSR,W
01CD 1803          00646          BTFSC   STATUS,C
01CE 29C9          00647          GOTO    Loop_C2
01CF 0184          00648          CLRF    FSR            ;Lastly clear FSR reg
01D0 0008          00649          RETURN
                   00650
                   00651 ;******          TMR0 IRS - Set Time_Bit for background tasks
01D1               00652 Do_Inc_Time
01D1 16A6          00653          BSF     FLAGS,Time_Bit ;Tell background tasks of overflow
01D2 110B          00654          BCF     INTCON,T0IF    ;Clear for next overflow
01D3 0008          00655          RETURN
                   00656
                   00657 ;******          Handle Interrupts Here
01D4               00658 Interrupt
01D4 00A2          00659 PUSH:    MOVWF   Temp_W
01D5 0E03          00660          SWAPF   STATUS,W
01D6 00A3          00661          MOVWF   Temp_Stat
01D7 0804          00662          MOVF    FSR,W
01D8 00A4          00663          MOVWF   Temp_FSR
01D9 1283          00664          BCF     STATUS,RP0     ;Point to BANK 0 - Very IMPORTANT !!!!!
                   00665
01DA 188C          00666          BTFSC   PIR1,TMR2IF
01DB 2005          00667          CALL    Do_OState      ;Go Do Task #1 - all states
                   00668
01DC 18A5          00669          BTFSC   T_B,IState1_B  ;INTF will set even if INTE is cleared
01DD 29E0          00670          GOTO    I1
01DE 188B          00671          BTFSC   INTCON,INTF
01DF 2097          00672          CALL    Do_I1State     ;Go Do Task #2 - 0 state only
01E0               00673 I1:
01E0 180C          00674          BTFSC   PIR1,TMR1IF
01E1 2097          00675          CALL    Do_I1State     ;Go Do Task #2 - 1-B states
                   00676
01E2 190B          00677          BTFSC   INTCON,T0IF
01E3 21D1          00678          CALL    Do_Inc_Time    ;Go Inc Time_Bit every 512uS
                   00679
01E4 0824          00680 POP:     MOVF    Temp_FSR,W
01E5 0084          00681          MOVWF   FSR
01E6 0E23          00682          SWAPF   Temp_Stat,W
01E7 0083          00683          MOVWF   STATUS
01E8 0EA2          00684          SWAPF   Temp_W, F
01E9 0E22          00685          SWAPF   Temp_W,W
01EA 0009          00686          RETFIE                 ;Return from Interrupt
                   00687
                   00688          END
```

5

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)
0000 : X---XXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX----- ----------------

All other memory blocks unused.

Program Memory Words Used:   488
Program Memory Words Free:  1560


Errors   :     0
Warnings :     0 reported,      0 suppressed
Messages :    19 reported,      0 suppressed
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN588

PICmicro™ Microcontroller Oscillator Design Guide

Author: *Kim Peck*
Consultant

Designing a clock oscillator without some knowledge of the fundamental principals of acoustic resonators is possible but fraught with the uncertainty of "cut and try" methods. While the oscillator may be made to run with the chosen resonator, it is quite likely that the unit will be slightly off the intended frequency, be grossly off frequency because it is operating on an unintended mode, or have unacceptable temperature characteristics because the wrong resonator was chosen for the application.

This application note is primarily for informational purposes. It is intended to help the designer of clock oscillators understand the parameters of crystal resonators and the terminology of the crystal resonator industry, both of which tend to be somewhat mysterious and arcane to the uninitiated. Details of crystal cuts and rotations, for instance, are of no use to the oscillator designer, only the designer of crystals. The oscillator designer still needs to understand and be able to predict the performance and trade-offs associated with these parameters. This is not an in-depth or rigorous treatment of acoustic resonators, but a practical guide, which should allow the designer to gain a basic understanding, and to help in choosing and specifying resonators.

INTRODUCTION

The main purpose of the oscillator in PICmicro microcontrollers, or almost any other microcontroller, is to provide a reliable clock for the controller processes. At the most basic level, the clock provides a timing interval to account for circuit rise times and to allow data to stabilize before that data is processed. This is a "synchronous" process. The clock also provides an opportunity for the programmer to perform time keeping of several types. In the PICmicro, the clock also drives hardware dedicated to timekeeping. The applications may include keeping "real time", or timing sensitive processes such as serial data communication. The accuracy of these timing applications is dependent upon the accuracy of the clock oscillator.

Design Challenges

The PICmicro microcontrollers offer unique design challenges because they are uniquely flexible. Flexibility usually demands difficult decisions on the part of the designer, but offers otherwise unattainable performance. The multiple oscillator options and wide range of operating voltages require awareness of advantages and trade-offs of various configurations. The PICmicro designer must be able to accurately predict stability performance of various configuration and then obtain that performance from the PICmicro clock in order to successfully implement these functions.

Wide Voltage Range

The PICmicro operates over such a wide voltage range that the oscillator parameters may be the limiting factor in the operation of the controller. If low power operation at low voltages is desired, the loop gain must be raised in order to insure reliable clock operation. If a nominal supply voltage is available, the loop gain must be reduced in order to prevent excessive power dissipation in the crystal. If battery operation is intended, then a careful balance must be struck between reliable operation at the low voltage, and damaging delicate resonators, or spurious oscillations at the high voltage when the battery is fresh.

Low Power

The outstanding performance of the Low Power option places a burden on the designer who would take advantage of this feature. The frequency chosen must be the lowest practical. Attention must be paid to the reactances associated with the crystal so as not to excessively load the oscillator output and cause excessive power consumption.

Low Cost

The low cost of the PICmicro series presents a challenge in finding commensurately low-cost components to complete the design. The relationship between cost and performance when various types of resonators are considered, is far from linear. The low cost of PICmicro microcontrollers, may remove them from the position of being the cost driver in some designs, challenging the designer to aggressively seek cost reductions in components which were previously not considered. The second challenge offered by such economical parts is that of new applications which were not considered practical before the advent of PICmicro processors.

© 1997 Microchip Technology Inc.

THEORY OF OSCILLATORS

Conditions necessary for oscillation

An oscillator is a device which operates in a closed loop. This condition can be difficult to analyze, but the techniques for analysis are as valid for motor speed controls as it is for phase lock loops and oscillators. Oscillators are somewhat unique in that they are intentionally unstable, but in a controlled manner. In order for oscillation to occur in any feedback system, two primary requirements must be met. The total phase shift must be zero or 360° at the desired frequency and the system gain must be unity or greater at that frequency.

The Ideal Oscillator

The ideal oscillator has a perfectly flat temperature coefficient, is 100% power efficient, has no limits on operating frequency, has no spurious modes, has a perfect output wave shape, and is available in the high degrees of miniaturization which exists in semiconductors. This oscillator of course, does not exist. The primary limiting factor for most oscillator parameters is the resonator. The following is a discussion of the trade-off, potential advantages and primary disadvantages of several popular types of resonators, and how they will behave in a PICmicro oscillator design.

RESONATOR BASICS

There are several types of resonators available to the designer of microprocessor clocks. They all provide trade-offs between performance, size, frequency range and cost. Resonators for clock oscillators usually fall into two basic groups. These are quartz and ceramic resonators. Historically, ceramic resonators came into use in oscillators much later than quartz crystals and derive all of their terminology and conventions from the longer history of quartz crystals. A third type of clock oscillator is the RC (resistor / capacitor). This oscillator is a relaxation type, and employs no resonator as such. While this type requires the same basic conditions for oscillation to occur it is better described using different techniques and analogies.

Quartz Resonators

Quartz is the crystalline form of silicon dioxide. This same material, in amorphous form, is commonly found as beach sand and window glass. As a crystal, it exhibits piezoelectric effects as well as desirable mechanical characteristics. A quartz crystal resonator is an acoustical device which operates into the hundreds of MHz. Its resonance and high Q are mechanical in nature, and its piezoelectric effects create an alternating electrical potential which mirrors that of the mechanical vibration. Although it is one of the most common of naturally occurring crystals, natural quartz of sufficient size and purity to be used in the manufacturing suitable resonators, is unusual and expensive. Almost all modern resonators are manufactured using cultured quartz, grown in large autoclaves at high temperatures and pressures.

Whether naturally occurring or cultured, quartz crystals occur as six-sided prisms with pyramids at each end. This raw crystal is called a "boule". In an arbitrary coordinate system the Z, or optical axis, runs the length of the crystal, connecting the points of the pyramids at each end. If one views this hexagonal bar on end, three lines may be drawn between each of the six opposing corners. These are called X axes. Perpendicular to each X axis is a Y axis, which connects opposite pairs of faces. When the boule is cut into thin plates or bars called blanks, the cut of the saw is carefully oriented either along, or rotated relative to one of these axes. Orientation of the saw is chosen based on the mode of vibration for which the plate is intended, and the desired temperature profile. Plates are usually rounded into discs. Types of crystal cuts are named for the axis which the cutting angle is referenced when the blanks are cut from the boule. After being cut and rounded, the blanks are lapped to frequency and any surface finishing or polishing is done at this time. Electrodes are deposited on the blanks by evaporation plating, and the blank is mounted in the lower half of the holder. It is finished to the final frequency by fine adjustments in the mass of the electrode plating, either by evaporation or electroplating. The top cover is then hermetically sealed by one of several methods, which include cold welding and solder sealing.

Most crystals made today are A-T cut, which employ a thickness mode. This mode provides the highest frequency for a given thickness of the plate, and the best possible frequency stability over most temperature ranges. Many other modes of vibration are possible. Flexure modes are usually bar shaped, and are used for low frequency (near 100 kHz) resonators. Tuning fork crystals are a special case of this type.

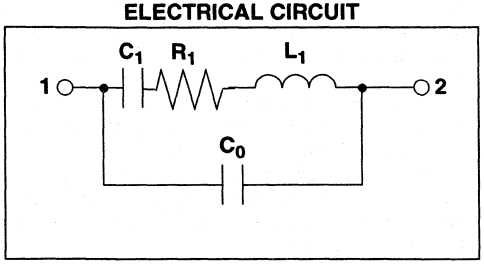
© 1997 Microchip Technology Inc.

Ceramic Resonators

Unlike quartz resonators, which are cut from a single crystal, a ceramic resonator is molded to a desired shape instead of grown. The material is polycrystalline form of barium titanate, or some similar material. The electrical model is almost identical, with the addition of one resistor, as the material is intrinsically conductive. The material is artificially made to exhibit piezoelectrically active by allowing it to cool very slowly, as in growing a quartz crystal (not nearly as long a time), but in the presence of a strong electric field. The molecular electric dipoles align themselves with the applied electric field. When the material has cooled, the alignment of the electric dipoles is retained, which is equivalent to piezoelectricity.

These materials have elastic properties that are not as desirable as quartz, and so their performance is not equal to that of quartz resonators. Specifically, ceramic resonators have far lower Qs and frequency deviations due to temperature on the order of 1000 to 10000 times greater than that of an A-T cut quartz crystal. The cost of ceramic resonators is much lower however, because the material is not grown under the extreme and expensive conditions that are necessary for quartz. They are also much smaller than A-T cut quartz resonators, particularly at frequencies under 2 MHz.

FIGURE 6: RESONATOR EQUIVALENT ELECTRICAL CIRCUIT



Since the "Q" of ceramic resonators is generally lower than quartz, they are more easily pulled off frequency by variations in circuit or parasitic reactances. This is desirable if a circuit is designed with a variable element, as greater tuning range is realized. It is not desirable if the highest possible stability is the design goal, because the resonator will be more susceptible to variation in parasitic reactances, such as capacitors formed by circuit board etch, and temperature variations of intended circuit reactances. These variances will add to the already substantial deviation over temperature of the resonator itself. If your stability needs are modest however, ceramic resonators do provide a good cost / performance trade-off.

Equivalent electrical circuit

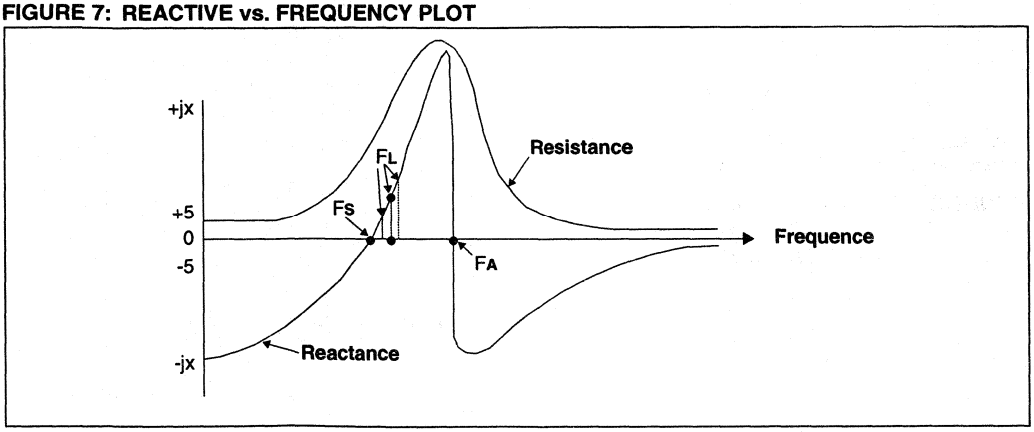
The circuit shown in Figure 6 is a close approximation of a quartz or ceramic resonator. It is valid for frequencies of interest to the PICmicro designer. Not all of the parasitic elements are shown as they are not important to this discussion. In this circuit, L_1 and C_1 are the reactances which primarily determine the resonator frequency, while a series resistor represents circuit losses. A shunt capacitor, C_1 represents the holder and electrode capacitance.

Because L_1 and C_1 are associated with mechanical vibration of the crystal, these are commonly referred to as motional parameters, while C_1 is called the static capacitance. The reactance of L_1 and C_1 are equal and opposite at the series resonant frequency, and their magnitude is very large as compared to R_1. The phase shift at the series resonant frequency is zero, because the reactances cancel. The series resonant frequency is calculated as shown in Equation 1.

EQUATION 1: SERIES RESONANT FREQUENCY

$$F_s = \frac{1}{2\pi\sqrt{L_1 C_1}}$$

FIGURE 7: REACTIVE vs. FREQUENCY PLOT



The actual series resonant frequency as determined by the zero phase point is slightly lower than this calculation because of the effects of C_0, and for practical purposes can be considered identical. This fact may be useful to those designing tunable crystal oscillators. These resonator parameters are generally considered to be constant in the region of the main resonance, with the exception of R_1. A plot of reactance over frequency is shown in Figure 7. The point labeled F_S is the series frequency, while F_L, is the frequency where the crystal is resonant with an external load capacitor. Operation at this point is sometimes called parallel resonance. F_A is the frequency where the crystal is anti-resonant with its own electrode capacitance. Only the region below F_A is useful as an oscillator. Notice that the resistive component begins to rise, before F_S and continues steeply above F_S. This makes operation with small load capacitors (large reactances) difficult. One must be sure that if the resonator is specified to operate at a load capacity that the maximum value of R_1 is specified at that operating point. The zero phase shift point is the most common method of identifying the exact series resonant frequency. When the series frequency is known, operation at a load reactance is easily calculated as follows:

EQUATION 2: OPERATION AT A LOAD REACTANCE

$$\frac{\Delta F}{F_S} = \frac{C_1}{2(C_0 + C_L)}$$

where ΔF is the deviation from F_S to F_L, F_L is the operating frequency when in series with a load capacitor, F_S is the series resonant frequency (without any load capacitor), and C_L is the load capacitor.

The value of R_1 at the frequency F_L can be approximated by:

EQUATION 3: VALUE OF R_1 AT THE FREQUENCY F_L

$$R_1 = R_L\left(\frac{C_L}{C_0 + C_L}\right)2$$

The reactance slope in the region of the series resonance can be approximated by:

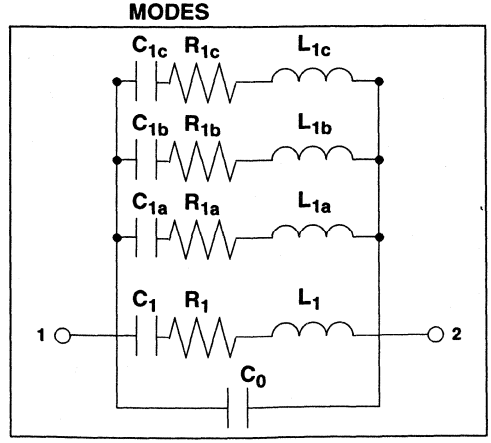
EQUATION 4: REACTANCE SLOPE IN REGION OF SERIES RESONANCE

$$\frac{\Delta X}{\left(\frac{\Delta F}{F}\right)} \approx \frac{10^6}{\pi F C_1}$$

where ΔX is the reactance difference, in Ω, from series, at which of course the reactance is zero. $\Delta F/F$ is the fractional frequency deviation from series resonance. F is the frequency of interest in MHz, and C_1 is the crystal static capacitance of Figure 6. This is only accurate in the region of series resonance and the accuracy declines as frequencies further away from series are considered. This parameter is useful in determining the optimum C_1, which the designer might specify in order to have the correct tuning sensitivity for any frequency adjustments, or given a crystal C_1, what tuning sensitivity will result from various reactive components.

The ratio of the reactance of L_1 or C_1 to R_1 is arbitrarily designated as Q. This is also known as quality factor, and applies to any reactive component. The series resonant frequency of the crystal is the sum of the total series reactances. Quartz A-T cut crystals exhibit spurious modes which are always found at frequencies just above the main response. These are always present and are not associated with activity dips. There are also odd ordered mechanical overtone modes. Any of these modes (spurious or overtone) can be modeled as duplicates of the primary RLC electrical model, and placed in parallel with it (Figure 8). Notice however, that there is only one C_0. Near the resonance of each series circuit, the effects of the other resonances may be considered negligible. Each resonance of course, has its own motional properties, the one of primary interest here is the R_1 of each resonance. The R_1 usually increases with increasing overtones, making higher overtones more lossy. The PICmicro designer must take care to specify the crystal spurious to always be of higher resistance than the desired response. This can be achieved in a well designed resonator. A heavy metal such as gold, as an electrode, will discourage higher overtones, by virtue of its higher mass. Crystals designed for high frequencies, almost always use a lighter material, such as aluminum. Electrode size also plays an important role.

FIGURE 8: EQUIVALENT CIRCUIT FOR SPURIOUS AND OVERTONE MODES



© 1997 Microchip Technology Inc.

OSCILLATORS

Phase and Gain

As stated earlier, two conditions must be met for oscillation to occur. The phase shift must be zero or 360° at the desired frequency, and the total system gain must be one greater or at that frequency. Logic gates or inverters are convenient for this purpose. They have large amounts of gain, limit cleanly, produce square waves, and their output is appropriate for directly driving their respective logic families. Most oscillators in this family use an inverting amplifier, as shown in Figure 13. The phase shift is 180° through the gate, and the two reactances at either end of the crystal are chosen to provide an additional 90° each, bringing the total to the required 360°. The primary effect of changes in phase is to shift the operating frequency (to tune the crystal). The primary effect of changes in gain is to cause the oscillator to cease functioning when reduced, or cause spurious modes and excess power to be dissipated in the crystal when increased.

Oscillation will occur at the frequency for which the total phase shift is 360°. This is true for any frequency (or resonator response) for which the gain is greater than unity (including unwanted responses). The series resistor (R_S) is used to adjust the loop gain, and to provide some isolation from reactive loads for the amplifier. The lower limit of loop gain is determined primarily by the need for sufficient excess gain to account for all variations, such as those caused by temperature and voltage (not just in the amplifier, the crystal resistance may change as a function of temperature). The upper limit of loop gain should be that where it becomes possible (or at least likely) for the oscillator to operate on a spurious mode. In some resonators damage to the resonator is the overriding concern regarding drive level. If the stability requirement is rather "loose" the stability problems may not be the first indications of trouble. Excessive drive levels in tuning fork types for instance, may cause damage to the point that the crystal unit fails. It is important to estimate drive levels before operation begins, include and adjust a series resistance appropriately, and by measurement, verify the results.

Estimating Drive Levels

The drive levels may be estimated with the following steps. First find load impedance presented by crystal network, including phase shift capacitors and amplifier input impedance. This is found by the following:

EQUATION 5: LOAD IMPEDANCE

$$R_n \cong \frac{X_c^{\,2}}{R_S + R_{OSC1}}$$

where R_N is the network impedance. X_C is the reactance of one phase shift capacitor (assuming they are the same). R_{OSC1} is the input impedance of the OSC1 pin (should include reactance). R_S is reactance + resistance at operating frequency ($R_S + X_S$).

The current delivered into this impedance is found by:

EQUATION 6: CURRENT DELIVERED

$$I_n \cong \frac{V_{OUT}}{R_S + R_N}$$

where I_N is the RMS current drawn by the network. V_{OUT} is the OSC2 output RMS voltage. R_N is calculated above. R_S is described above. The current which passes through the crystal then is found by:

EQUATION 7: CURRENT THROUGH CRYSTAL

$$I_S \cong \frac{V_C}{R_S + R_{OSC1}}$$

The power dissipated by the crystal is then found by I_S squared times the crystal R_1.

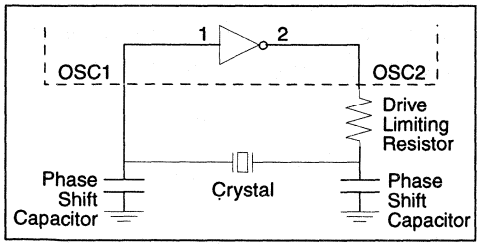
Controlling Drive Levels

When designing any oscillator, one should take care not to lower the loaded Q of the resonator by inserting any resistive components between the phase shift capacitors (or any other reactive components) and the crystal. If It is necessary to reduce the drive level to the crystal, or lower overall loop gain, resistance should be inserted between the amplifier output, and the crystal (Figure 9). This method is much better than changing load reactances, which will have no significant effect on gain until the frequency has been pulled well away from the design center. This will also have the more significant effect of raising operating current, because if no series resistor is present, larger reactance of the phase shift capacitor will load the OSC2 output directly. If a very low drive level is required, such as with tuning fork type crystals, the series resistor is the best method. The resistor should be adjusted until the unit just runs with a typical crystal at the lowest operating voltage, and resulting drive measured at the highest operating voltage. The actual resistor value is best determined experimentally with a representative sample of crystals, and a broad range of values should be satisfactory. In general, the point where oscillation stops for any crystal unit (within specified parameters), is the resistor's upper limit. The lower limit may be 0 Ω, for a less fragile crystal type, depending on the operating frequency. If no spurious or overtone modes are encountered, it is likely that the oscillator may have relatively little excess

© 1997 Microchip Technology Inc.

gain at that operating frequency. If resulting drive level at higher voltage is still unacceptable, then supply voltage variations must be reduced.

FIGURE 9: PICMICRO OSCILLATOR CIRCUIT



Measuring Drive Levels

Drive levels cannot be easily measured with any certainty by reading voltages at each end of the crystal. This is because of the phase shift which is present in varying degrees, depending on how close to series resonance of the crystal, the oscillator is operating. It is much more reliable and accurate to measure the crystal current with a clip-on type oscilloscope current probe. This probe may require an outboard amplifier in order to measure very low drive levels. It is also important to accurately know the series resistance of the crystal under the same operating conditions of frequency and drive level. This information is easily obtained with a network analyzer or a modern crystal impedance meter. While the oscillator designer may not be equipped with such a meter, the manufacturer of the crystal most certainly should be, and resistance data should be provided for at least one, and perhaps several possible drive levels, if variations in drive are expected.

UNDESIRED MODES

Mechanical resonators are not perfect devices. They exhibit many spurious responses, either continuously or over narrow temperature ranges. If a quartz resonator is swept with a R.F. network analyzer, several smaller responses will be seen just above the main response. These are always present in mechanical plate resonators. For oscillator applications, they must be specified to have a lower response than desired mode. The crystal designer can control these to some extent by varying plate geometry and electrode size. These spurious modes are usually similar in nature to the main response, and do not vary in relation to it to any important degree. Other spurious are caused by completely different modes of vibration, and have radically different temperature curves. These may lay unnoticed until a temperature is reached where the two temperature curves intersect. At this one temperature, the spurious mode traps some of the mechanical energy created by the main mode. This causes a rise in the series resistance, usually accompanied by an unacceptable change in frequency. With a very small change in temperature, the effect will disappear. This is know as an "activity dip", activity being a dimensionless mechanical property which is inversely proportional to resistance. These can also be successfully specified away in most resonators. Any response of the resonator, be it from spurious, or mechanical overtones, may control the oscillator output frequency if phase and gain criteria are met. In some unusual circumstances, the oscillator may run simultaneously on two or more modes. In general, the fundamental response of any mechanical resonator is usually the largest (lowest loss), and the oscillator will run on this response if no other circuit elements are introduced which favor higher frequencies. If the desired frequency is such that the third overtone, begin the first available (mechanical overtones are always odd ordered), is below 15 or 20 MHz, the oscillator may occasionally run at around three times the desired frequency. This may only happen every third or fifth time the unit is activated. The unit may start correctly, but jump to higher overtone when the unit is exposed to a very narrow temperature range, but remain there after the temperature has changed. The best fix for this problem is usually a reduction in overall loop gain. Occasionally a crystal may have a very low resistance at overtone modes as well as the fundamental. In this case it may be useful to specify overtone modes, as spurious and guarantee at least a -3dB difference between the overtone and the fundamental responses. This condition will already exist for 99% of the resonator designs, and is not usually specified.

It is also best not to insert any large reactances which would compete with the Q of the crystal for control of the oscillator output frequency. If this is done (say, for the purpose of adjusting the oscillator frequency), the tuning reactance (usually a variable capacitor) must be accompanied by an equal reactance of the opposite sign in order to bring the total loop reactance back to zero (unless the crystal is designed to operate with that large series reactance, which could cause other problems). If the oscillator is pulled far enough from the series frequency, the rising crystal resistance will lower the loaded Q of the crystal until the reactance slope of these components competes with that of the crystal. This will cause the oscillator to "run" on these components instead of the crystal, the loop being completed by the C_0 of the crystal. The component with the steepest reactance slope will control the frequency of the oscillator. The tuning sensitivity of these components will also be directly proportional to the magnitude of their reactances. Any unwanted variation of these components will have increased consequences for the stability of the oscillator. Another source of spurious is a relaxation mode which is caused by the amplifier bias circuits and the phase shift capacitors. The loop is completed through the crystal C_0. Again, a series resistor will usually solve this problem, although in some cases the amplifier bias values may need to be changed.

© 1997 Microchip Technology Inc.

Load Capacitors

In gate or logic type oscillators, the crystal is usually manufactured to be slightly inductive at the desired frequency, and this inductance is canceled by the two phase shift capacitors. The primary purpose of these capacitors is to provide the phase shift necessary for the oscillator to run. Their actual value is relatively unimportant except, as a load to the crystal, and as they load the output when no series resistor is used. These reactances are the sum total of selected fixed capacitors, any trimmer capacitors which may be desired, and circuit strays. If a loop is considered from one crystal terminal through one phase shift capacitor through ground and the second phase shift capacitor, to the second crystal terminal, all the reactances including the crystal motional parameters must add up to zero, at the desired operating frequency.

As a crystal load, all circuit reactances external to the crystal should be thought of as a series equivalent. In order to know the total load reactance seen by the crystal, the total shunt reactances on either terminal are summed, and the series equivalent is calculated. This should include the OSC1 and OSC2 terminal reactances, but these are negligible if they are sufficiently small when compared to the phase shift capacitors. The value of these capacitors, is then chosen to be twice the specified load capacity of the crystal. It some adjustment of the frequency is necessary, one of the phase shift capacitors can be chosen at a smaller value, and the difference made up by a variable capacitor placed across it. An alterative method is to place a larger value trimmer capacitor in series with the crystal. The value of the trimmer capacitor must be chosen along with the phase shift capacitors, all in series, to give the correct load capacity. Frequency should not be adjusted by shunting the crystal with a capacitor. If it is desired to use a crystal which is finished at series resonance, an inductor of equivalent reactance to half of the phase shift capacitor, must be placed in series with the crystal.

STABILITY

General

Frequency stability is the tendency of the oscillator to remain at the desired operating frequency. Its deviation from that frequency is most conveniently expressed as a dimensionless fraction, either in parts per million (PPM) or a percentage. Absolute deviations in Hz must always be referenced to the operating frequency, which is less convenient and not universal. In the following discussion of temperature characteristics, one can see that the fractional deviations are universal without any direct effect of operating frequency. In order to calculate a total frequency stability, various separate elements must be identified and quantified. Not all parameters of frequency stability are important to each design. The various items which effect the frequency of an oscillator are: the temperature profile of the resonator, the resonator's room temperature frequency tolerance (also known as "make tolerance"), its long term frequency drift which is normally know as ageing, and its sensitivity to other circuit reactances. Is it possible to adjust it to the exact desired frequency? If not, how big is the error due to other component tolerances. Due to the complexity of this combination, most crystal manufacturers will offer a standard crystal which is guaranteed to be ±100 PPM over -20°C to +70° C, or ±30 PPM over -0°C to +60°C. Note that the temperature coefficients of some of the curves in Figure 7 are much smaller than this over the same temperature range. Large portions of these tolerances are devoted to make tolerances and circuit component tolerances. The room temperature items can be relatively simple to specify in the resonator design. If careful attention is paid to specifying the crystal, or designing the oscillator to accommodate a standard crystal, more of the total stability requirement can be devoted to the temperature profile, or the overall stability requirement can be reduced. The temperature profile, however, is subject to other circuit influences external to the resonator. These may be somewhat more difficult to perceive and control. If, for example, the chosen resonator is an A-T cut or tuning fork type, possessed of a nominal temperature profile of less than 50 PPM over the desired temperature range, external influences, such as capacitor temperature coefficients, may play an important part in the overall stability of the oscillator. If however, a ceramic resonator is chosen, it's temperature profile of 40 to 80 PPM/C will dominate the oscillator stability, and 5 or 10 PPM shift from changes in amplifier impedance or capacitor temperature coefficients will not be important. The designer may choose a crystal even when the overall stability specification (of the oscillator) does not require it, giving large design margins. If any amount of testing or adjustment of the oscillator frequency is needed with the lower cost resonator, the crystal may be more cost effective. When designing any resonator as part of a simple logic type oscillator circuit (Figure 9), some attention should be given to swapping the amplifier reactances (that is to make them a very small part of the sum total circuit reactance) with the phase shift capacitors, and any other circuit reactances. This is at least, a good design practice. The largest reactance has the most effect on the operating frequency. It follows then that the motional parameters, which have very large reactances, dominate the equation for the total reactance, and so the operating frequency of the oscillator.

Another good design practice, is to specify only as much pullability as is required to accommodate the make tolerance and ageing of the resonator, and tolerance of other circuit elements. Pullability is a function of the ratio of C_1 to C_0. As the reactance of the crystal C_1 increases it becomes more stable in relation to outside reactive influences. It also becomes more difficult to intentionally adjust its operating frequency. If too high a C_1 is specified, the resonator will be sensitive to external influences, and the effect of these influences may be as large or larger than the temperature profile. If the

© 1997 Microchip Technology Inc.

C_1 is to small, it may not be possible to adjust the unit exactly to the desired operating frequency. The small electrode size needed to realize a low C_1 may also concentrate the mechanical energy in a very small percentage of the blank, causing unpredictable behavior. In order to quantify pullability in terms of C_1 to C_0 ratio and load capacitance (refer to the Equivalent Electrical Circuit section).

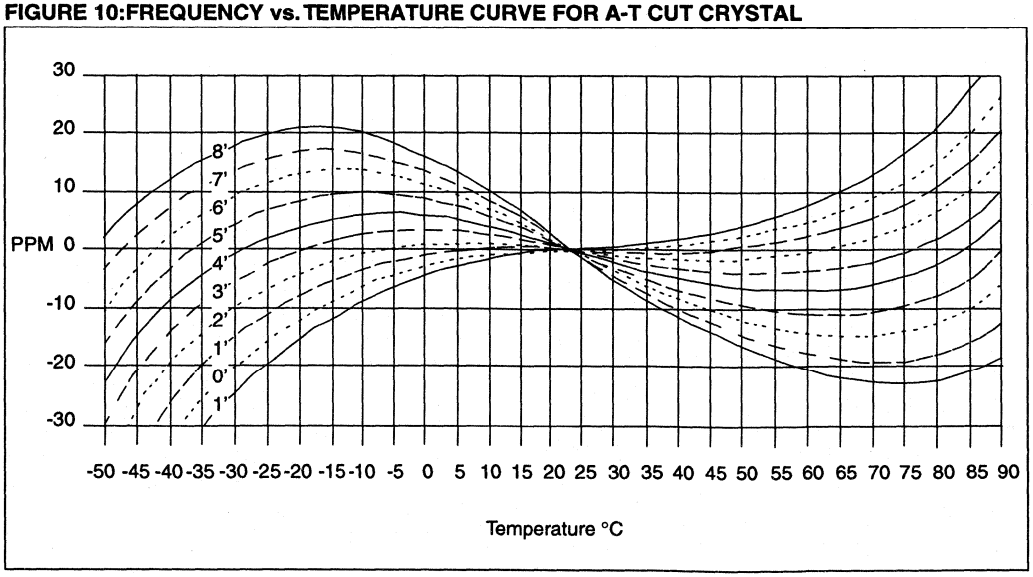
A-T Cuts

The A-T cut crystal and its variations, is by far the most popular resonator in the world today. A-T cut crystals are popular because the "S" shaped temperature curve is centered very near room temperature, typically around 27°C. This temperature profile is compact, symmetrical, and most manufacturers are able to provide good control of the cut angle.

Because most of the crystals manufactured in the last 40 years have been A-T cuts, they are very well understood and documented. This is important because while temperature coefficients can be calculated from the mechanical properties, such as elastic constants, they can (and have been) measured with much more accuracy. When the temperature coefficients are accurately known, the temperature profile can be calculated for an individual set of conditions. Figure 10 is a family of temperature curves of A-T cut crystals used for this purpose. Each curve represents a possible crystal at incremental changes in the cut angle. The practical limit for accuracy of the cut is about ± 1 minute of angle, and in any lot of crystals there will be variations of about ± 1 minute. The designer will create a box around these curves using the desired temperature limits as the vertical sides, and the desired frequency tolerance for the horizontal lines, as shown in Figure 10. If the curves are spaced at intervals of one minute of angle, then the specification is a practical one if three of these curves (± 1 minute) fit within the outlined area. It is possible to purchase crystals with a closer tolerance, but this is mostly a matter of yields, rather than a better process. The steeply increasing cost will reflect the higher reject rate.

When purchasing a crystal, do not attempt to specify a specific angle, rather specify a frequency deviation between turning points, with tolerances. The mathematics of these curves, is represented by a linear term between two turnover points, whose inflection point is at or near 27°C. The temperature above the high turnover and below the lower turnover, are characterized by cubed terms (very steep). This was described by Bechman in the late 1950s as a third order polynomial. This can be seen in Appendix A. Notice that as the linear portion of the curves between turnover points approaches zero slope, the turnover points move closer together. This tends to limit the temperature range over which very small stabilities can be realized. If the required operating temperature range is inside of the range of the turnover points, a low angle is desirable. If so specified, most manufacturers will provide a crystal with temperature profiles on the order of ±5 to ±10 PPM over modest temperature ranges for a reasonable cost. If the desired operating temperature range is outside of the range of turnover points, a higher angle is desirable in order to keep the frequency at extreme temperatures within the same realm as deviation between turnover points. This may approach ±60 PPM for large temperature ranges, but is still far less than the smallest deviations achievable with other resonators over the same temperature range.

FIGURE 10:FREQUENCY vs. TEMPERATURE CURVE FOR A-T CUT CRYSTAL



© 1997 Microchip Technology Inc.

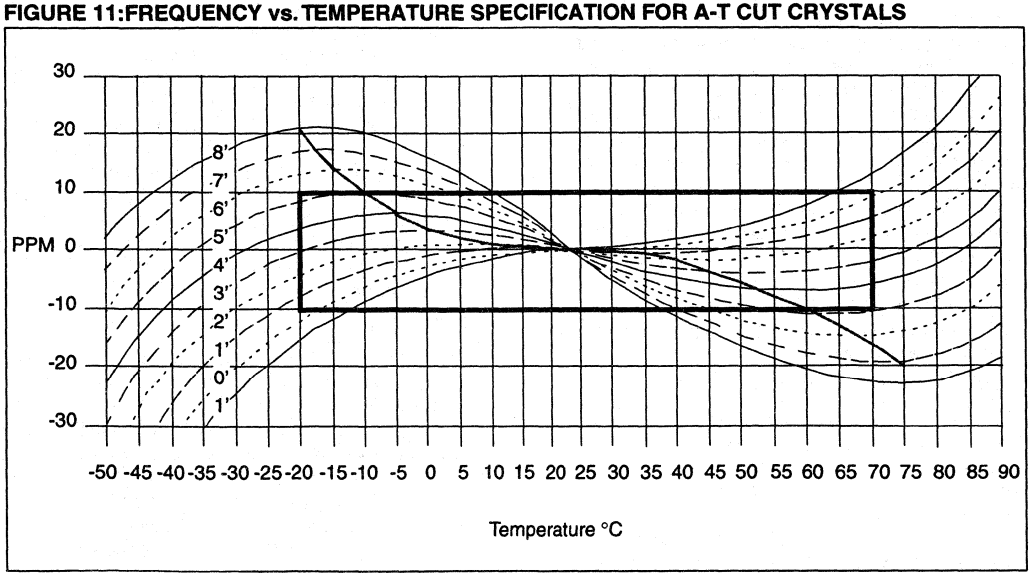
What is not immediately obvious is that if a linear frequency shift with temperature is applied to a frequency curve the result is a rotation of the curve which will eventually match another member of the curve family. There is no other distortion of the temperature curve if the frequency shift is linear, such as from a temperature compensating capacitor. This fact also gives a convenient graphical technique to estimate the effect of temperature coefficients of other components. There exist several flaws in this picture of the A-T cut temperature profile, which may prevent the PICmicro designer from completely realizing the stability suggested by the curves in Figure 10.

The first problem which may arise when choosing a crystal angle based upon these curves, is that there may occur some rotation of the crystal angle due to external circuit influences. The most common influences are that of reactive components (inductors and capacitors). Most inductors have a slight positive temperature coefficient, while capacitors are available in both positive and negative temperature compensating types. Non-compensating type capacitors vary greatly depending on the dielectric from which they are manufactured. The best capacitors for frequency determining elements, are ceramic types with NP0 (flat) temperature coefficients. Avoid at all cost, capacitors made from Z5U material. These have a large temperature coefficient and are unsuitable even for supply line decoupling or D.C. blocking capacitors. This is because a slight change in the R.F. impedance which shunts the VCC and VDD pins, will have an effect on the output impedance of the amplifier, and so an effect on frequency. The effect will be on the order of a few PPM, and may well be of secondary importance, depending on the stability requirement. A word about D.C. voltages and crystals. It is permissible to place a D.C. voltage across the terminals of the crystal. This does cause a small change in frequency, but that change is not significant for stabilities of ±5 PPM or greater.

The second problem is one of dynamic temperature performance. When the unit has stabilized at any temperature on the curve, the frequency will agree with the curve. While the temperature is slowing however, the frequency may be in error as much as 5 to 15 PPM depending on the temperature change. This effect is caused by mechanical stresses placed on the blank by temperature gradients. These can be minimized by thermally integrating the crystal, and joining it to a larger thermal mass. One oscillator engineer has been known to attach a block of alumina (ceramic) to both of the crystal pins in order to join them thermally. Any other mechanical stresses placed upon the pins or leads of an A-T cut crystal unit will also result in a dramatic frequency shift (if the unit is not damaged first). This is to be avoided.

5

FIGURE 11:FREQUENCY vs. TEMPERATURE SPECIFICATION FOR A-T CUT CRYSTALS



© 1997 Microchip Technology Inc.

The third item which will cause a deviation from the curves of Figure 10, is spurious response. This is known in the crystal industry as an activity dip. This name originates from a time when the series resistance was referred to as crystal activity, and the frequency change is accompanied by a marked rise in series resistance. This phenomenon occurs when mechanical energy is coupled from the normal thickness shear mode into another undesired mode of vibration. Several other modes are possible for finite plate resonators, and they will usually resonant at frequencies well away from the design frequency. These modes will often have radically different temperature profiles, and may intersect with the profile of the desired mode at only one very narrow range of temperatures (much less than 1°C). This makes an activity dip difficult to spot in normal testing. Those which are discovered are often around room temperature where temperature changes are more gradual. This coupling between modes is greatly effected by drive level, and the best crystal may exhibit a dip if grossly overdriven. Fortunately, most manufacturers today can produce a crystal which is free of significant dips if so specified. As the accompanying rise in resistance is occasionally large enough to cause oscillation to halt, the PICmicro designer should always specify activity dips to be less than 1 PPM, even if the overall stability requirement is much larger than this. In the interest of low cost and flexibility, the designer may also specify activity dip in terms of a maximum change in resistance.

The other important effect on frequency stability of A-T cut crystals, is ageing. This is the long term frequency shift caused by several mechanisms, the most notable being mass loading of the resonator, causing the frequency to shift ever downward. Because this is the primary mechanism, the cleanliness of the interior of the unit is of prime importance. This is in turn greatly effected by the method used to seal the unit, and the type of holder chosen. If the unit is subjected to excessive drive levels, the frequency may age upwards, indicating electrode material is being etched off of the blank. A good general purpose high frequency crystal using a solder seal holder may be expected to age about 10 to 20 PPM / year maximum. Resistance weld holders will average 5 to 10 PPM / year, and for high stability applications, cold weld crystals are available at ageing rates of 1 to 2 PPM / year. The ageing rates of most crystals will decay exponentially, the most change being in the first year. Ageing rates are different if the unit is operated continuously, but aging will continue even if the unit is not operated.

32 kHz Watch Crystals

The typical 32 kHz watch crystal is a tuning fork type. This is a special case of a flexure mode (N -T cut). The unusual nature of this flexure type is that it is indeed shaped like a tuning fork. This shape gives the crystal a very small size for its low frequency of operation and is almost always manufactured in the NC 38 holder. This is a tube 3 mm x 8 mm. This type is available at frequencies from 10 to 200 kHz, although 32.768 kHz is by far the most popular frequency. The frequency is of course 215, which is ideal for time keeping applications, and being so low is ideal for low-power applications. This type is generally less stable than higher frequency A-T types, but is much better than ceramic resonators, the primary attraction being the possibility of very low operating power drains. The PICmicro LP option was designed with this crystal in mind. It has a parabolic temperature profile of about .04 PPM / (°C) 2. The turnover point of the temperature profile is near 25°C. In order to calculate the change in frequency it is only necessary to square the difference in temperature from 25°C and multiply by .04. The temperature profile is shown in Figure 12. The C_1 is on the order of .002 pF, which will make design for frequency adjustment possible but not trivial. The make tolerance is usually about 20 PPM at best, making some adjustment necessary for most applications. The series resistance of this type is very high, on the order of 30,000Ω. It is imperative that care be taken to limit the drive to the crystal. Only a fraction of a mA of crystal current will damage this unit, possibly causing it to cease oscillation. This is best done with a series resistor between the OSC2 pin and the junction of the crystal lead and phase shift capacitor (Figure 12). If the frequency is moving upward in a continuous manner, the drive level is probably too high. A portion of this change will be quite permanent.

Ceramic Resonators

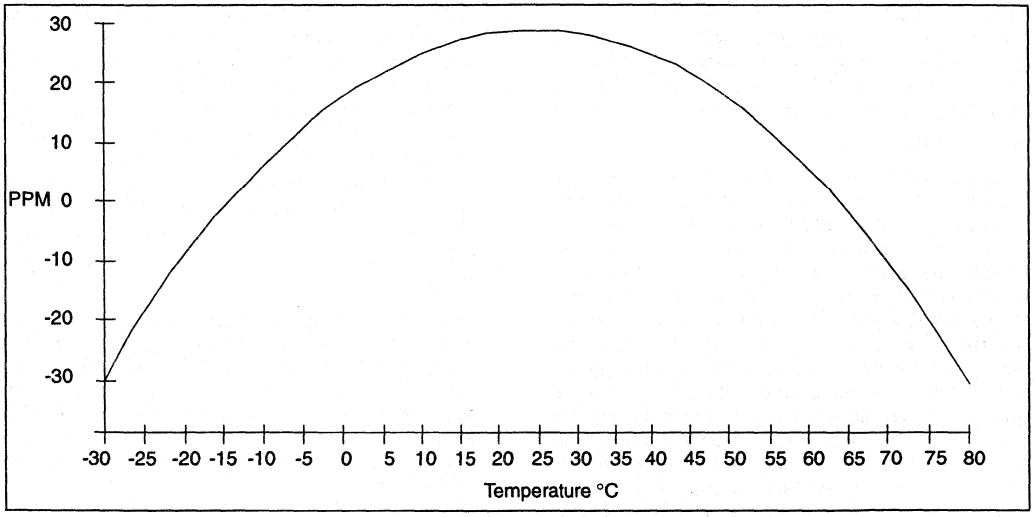
Ceramic resonators are the least stable type available other than the Resistor/Capacitor networks. The temperature profile is a much distorted parabolic function, somewhat resembling that of some capacitors. Temperature coefficient is on the order of 40 to 80 PPM /°C. Typical specified stability for -20°C to +80°C is ± 0.3% (3000 PPM). The C_1 can be as high as 40 pF, making the oscillator extremely vulnerable to circuit influences external to the resonator. The Rs however is on a par with A-T type crystals, at around 40Ω. The positive features of this type are the small size, low cost, and relative simplicity of designing it into a PICmicro part. Because these have a very low Q, the start-up time can be very good, although with the large phase shift capacitors necessary at low frequencies where this would be an advantage, the bias stabilization time will probably dominate the start-up characteristics. If the stability requirements are very modest, this will be a good choice.

R/C Oscillators

The PICmicro parts can be configured to operate with only a resistor and a capacitor as frequency determining elements. This is a very low cost method of clocking the PICmicro. The stability achieved this way is at best only adequate if the only thing required of the oscillator is to keep the PICmicro marching along to the next instruction. The main effects on stability are that of the switching threshold of the OSC1 input, and the temperature coefficient of the resistor and capacitor.

© 1997 Microchip Technology Inc.

FIGURE 12:FREQUENCY vs. TEMPERATURE FOR NC38 TIMING FORK TYPE CRYSTAL



HOW TO CHOOSE A RESONATOR

Type Trade-offs

The primary trade-offs for a designer when choosing a resonator are frequency, size, stability and cost. The lowest cost oscillator is the RC type. This also has the worst stability. The components however tend to be reliable and small, where as resonators are in general larger and have limitations on the amount of physical punishment they can absorb.

A-T cut crystals have the best overall stability and are available in frequencies from 1 MHz to the upper limit of the PIC16C5X part, and in a roughly 0.5 "square package". T-05s and 0.3 "square packages" are available at higher costs, down to a frequency of around 5 MHz A-T cut crystals also have a smaller overall temperature profile which the designer has the best chance of specifying and controlling. Temperature stabilities on the order of ±10 PPM are possible over modest temperature ranges. The A-T cut can be sufficiently reluctant to move off frequency, in response to parasitic reactance changes, that it can fully realize these small deviations over temperature. Such is not always the case with other resonator types or incorrectly specified crystals.

Ceramic resonators offer smaller size and slightly lower cost, although in large quantities, microprocessor grade crystals (±100 PPM) can be competitive. Ceramic resonators will, however, suffer from temperature stabilities in the 0.3% to 0.5% region. This is a significant step down from quartz crystals of any kind.

A designer must choose a resonator which is available in the desired frequency range, has acceptable temperature characteristics, has the lowest cost package which is appropriate for that resonator and is suitable for the mechanical packaging of the oscillator chosen. A-T STRIP resonators are normal A-T cut resonators in which the resonator blank is cut in a long strip rather than a disc, and the electrodes cover a much higher percentage of the quartz blank. A standard A-T cut crystal is a thickness mode resonator, and is usually cut in the form of a disk. The electrodes usually cover only a small portion of the blank. The remainder of the blank, not covered by the electrodes, can be thought of as support structure. By removing this support structure, the size of an A-T cut resonator can be greatly reduced. This type of construction violates several rules having to do with thickness to diameter ratios and greatly reduces the overall mass of the blank. This results in reduced performance in the form of slightly less predictable temperature stability, and dramatically reduced power handling capabilities. The A-T STRIPs are generally available up to 20 MHz, depending on the manufacturer.

Tuning Fork type resonators are a type of flexure mode resonators. They are made from quartz, are very small and available at a cost which is competitive with microprocessor grade A-T cut crystals. Tuning forks have a predictable parabolic temperature coefficient, but any drive power in excess of their very low specified level will deteriorate this quickly.

If stability requirements are beyond what is achievable with a good, A-T cut crystal, the next option is to drive the OSC1 pin with an external oscillator. A good Temperature Compensated crystal(X) Oscillator (TCXO) is expensive when compared to crystal resonators. Stabilities of ±1.0 PPM over large temperature ranges are common.

© 1997 Microchip Technology Inc.

Price discounts for volume quantities do not always occur, because each unit must be individually compensated. This varies greatly with the stability and temperature range, and so of course does the price, which in any case will be much higher than any resonator, which the PICmicro designer might consider.

Size and Performance

It is generally true that as a designer considers resonators of smaller size, he or she is faced with decreasing overall performance. Even the A-T cut crystal, which has the best stability discussed here, will become less stable as size is reduced, especially when plate area is reduced in relation to the spot as in a STRIP A-T. One important factor is the thermal inertia represented by the mass of the blank. The bigger the blank, the slower it is to follow the changes in temperature. When the blank changes temperature too quickly, it will deviate from the temperature profiles. The frequency will return to this point once the blank has stabilized at the new temperature, but may be well off the profile during a temperature slew. This problem becomes greater as the size of the blank is decreased. Some stability issues are due to the unusual motional parameters associated with certain miniature resonators. Some low frequency types have a C_1 which is larger than the holder capacitance (C_0), making it extremely easy to tune and vulnerable to external influences. Most miniature types have parabolic temperature coefficients which are large enough to make them inferior to A-T cut crystals as well as A-T STRIP cuts, and not only suffer from temperature transient problems as mentioned above, but it is also difficult to control their cut angle and finish frequency. Tuning forks have a somewhat more predictable if larger temperature profile. Almost all miniature types will not perform well (or sometimes at all) with excessive drive levels. The drive power to the resonator must be controlled, and usually one resistor is sufficient.

Cost and Performance

The lowest cost timing system, of course, is the RC type. This rugged, low cost and small timing system is useful only for the most forgiving timing applications. If all you need is something to keep the PICmicro moving, this is a good choice. The next step in cost is most likely the ceramic resonator. Its size is smaller than most A-T cut crystals, but its frequency stability is measured in percent, rather than PPM. A high C_1 also makes it vulnerable to external influences.

Tuning fork types, like all others, vary in price. They may cost less than the ceramic resonator, if a standard frequency is acceptable, or may cost more than an A-T cut if a nonstandard frequency is ordered. Tuning forks usually have a very low C_1 contributing to overall stability. This may actually make it difficult to trim to frequency. Tuning forks have a relatively controllable parabolic temperature curve. A-T cuts have the best overall stability but their cost varies greatly. A-T cuts have an additional advantage in that their temperature profile is the most easily controlled. This offers some flexibility in specifying the angle of cut. A low angle may be order for minimum deviation near room temperature, or a high angle may be ordered to give minimum deviation at extreme temperatures. Standard frequencies and loose specifications for motional parameters will yield cost and delivery competitive with ceramic resonators. Any nonstandard parameters will raise the cost quickly and almost certainly rule out any "off the shelf" part. It will be necessary to specify a nonstandard crystal, if the greatest possible stability is to be "wrung out" of an A-T cut crystal.

Packages

Quartz crystals have a large and mostly obsolete stable of resonator holders to choose from. This is because of the much longer history of quartz crystals. Most of the nomenclature used to describe them, and the technology used to develop them, comes from the MIL-STD system. These include the H/C 6, which is about .750" square, and is only necessary to accommodate the lowest frequency A-T cut. The H/C 43 is only about .500" square, and probably accounts for most of the crystal production in the world today. The H/C 45 is still smaller at about .350" square. There are many other standard part numbers which are variations of these, with pins or wire leads, thin version and short versions, and several different methods of sealing the package. Most manufacturers offer their own nonstandard variations of these, as well as clever ways to surface mount them. Most of these variations however, have their origins in the standard H/C parts. The method used to seal the package will have the greatest impact on price and ageing. Solder sealed crystals are usually the least expensive owing to the modest equipment requirements, and simplicity. Resistance weld is slightly more costly, and cold weld is a distant third. This may be changing as more large volume production is implemented with resistance welded packages. Because more exotic (expensive) materials are involved in the cold weld and resistance weld packages before any crystal is mounted in it, is doubtful that this order of cost will change very much. Both solder sealed and resistance welds leave some residue, which over long periods of time contaminate the blank. This causes long term frequency shifts, known as ageing. Cold welded crystals cost more because of more expensive materials which must be used, and expensive tooling (dies) which eventually ware out. Cold weld packages, if assembled in a clean environment, have the potential for the lowest ageing rates. Glass crystal holders have in the past held a slight advantage in ageing over cold weld types, but in the last several years, cold weld techniques have matured to where they have surpassed the glass holder in performance. Some manufacturers, because of the processes in place, may offer glass at a competitive cost. There is nothing wrong with glass holders, but these have no particular advantage over a modern cold weld package. In any case, the differences in ageing rates will not be important to all but a few PICmicro designers. Most ceramic resonators are only available in two or three packages, depending on the

© 1997 Microchip Technology Inc.

manufacturer. The most popular is the dip molded, ranging from 0.3" to 04" square, with some higher frequencies available in lower profiles. Their major size advantage over crystals, if any, will be in height rather than footprint size.

Design Examples

A communications device that is designed around the PIC16C5X part and requires that connecting units have a close timing relationship. Size is not a primary factor, but cost and stability are. A high clock frequency is desired in order to obtain a good sampling rate of the input signal. The PIC16C5X-HS part is selected for a clock frequency of 8 MHz. An X-T Cut crystal is chosen and specified for a maximum frequency deviation of ±40 PPM over -20° to +70°C. The frequency is too high for a tuning fork type, and the stability is out of the question for a ceramic resonator. An examination of A-T cut frequency deviation / temperature curves show that a ±1 minute angle tolerance gives ±30 PPM frequency deviation over temperature. This leaves 10 PPM for ageing over the products five-year life. An A-T STRIP is a choice but at this quantity, the A-T cut in an H/C 43 cold weld type holder, comes in at a lower bid. Since there is space in the assembly, it is chosen.

HOW TO SPECIFY A CRYSTAL

When the PICmicro designer chooses a resonator, whether it is a standard or a custom part, a specification, while not essential, is an extremely good idea. A clear specification covering all items of form, fit and function, will eliminate any possibility of confusion on the part of the manufacturer, and insure the part will be suitable for the application. The specification should communicate your requirements to the manufacturer and be an instrument by which questionable parts may be measured. Time for discussion with the manufacturer of the resonator is when the specification is being written, not after. The designer must have or gain knowledge of what parameters raise the difficulty level of manufacturing the resonator, and so the cost. Items which effect cost and levels at which these items become an issue, may vary between manufacturers. A typical crystal design sheet is shown in Figure 13. The A-T cut crystal is likely to have the most detailed specification. Other types of resonators will follow this general form with differences being mostly that of omitting many items. This data sheet is likely to become a document in a drawing package for design of a larger assembly, so the sheet begins and ends with blocks for a drawing number, sign-offs, and revisions. The title informs the manufacturer that the crystal is intended for use in an oscillator, as opposed to filters, or other applications.

Motional Parameters

The first item in crystal design is frequency and the operating load. This might include series resonance (no load), but the PICmicro designer will almost certainly use a value of about 1/2 of the phase shift capacitors, plus any trimmer capacitors which may be added. It is customary to use a standard value here such as 20 or 32 pF, but a nonstandard value is not very difficult given modern manufacturing equipment. The frequencies possible with the PICmicro oscillator should not strain the capabilities of most manufacturers.

The second Item is the "Make Tolerance". This the accuracy to which the crystal is manufactured at room temperature. This should be at least as small as the temperature deviation, and a ± 20 PPM should not effect the cost significantly. Avoid tightly specifying this value. Tolerances of ± 10 PPM and less are quite practical but more difficult and will impact cost. If the stability budget does not allow this for at least ± 20 PPM for tolerance of the crystal and associated components, then an adjustable component may be necessary. The added cost of parts and labor to adjust them must be weighed against the cost of tighter make tolerances on the crystal. This decision must be made on an individual basis.

The third item in the design parameters is the mode of vibration. This will be the fundamental mode for almost all PICmicro designers. Other possibilities include the third overtone operation, but many other parts must be added in order to insure operation on only the desired overtone. While there are some advantages to overtone operation, almost all PICmicro designers will specify the fundamental mode. Still, what may be obvious to the PICmicro designer must be conveyed to the crystal designer, and so this item should not be omitted, or minimized. Series resistance is usually a "not to exceed" value. A good fundamental mode crystal in the PICmicro operating frequency range will not be above 10 or 15Ω, although the oscillator may run with a higher value. This depends on the frequency and excess gain available from the particular model of the PICmicro part, at that frequency. The higher resistance will mean more power dissipated in the crystal, and for this reason a nominally lower value should be adhered to. The load capacity will have an effect on this value. The practically achievable series resistance will rise as the load moves the operating point away from series resonance and towards anti-resonance (Figure 12).

The motional capacitance, or C_1, may be the most troublesome item for the PICmicro designer to specify. This item will have the single largest effect on the tuning sensitivity (intended or unintended) of the oscillator. Additionally, if the C_1 is specified to be too small, the crystal designer, who controls *C1* by adjusting the electrode size, will use a very small electrode. This will result in the drive power being dissipated by a small portion of the crystal blank, making drive related areas, such as activity dips and other spurious, more critical. If a large C_1 is specified, the unit may be unnecessarily less stable. The static capacitance or C_0, is usually a "not to exceed" value, and it is not of much interest the PICmicro designer unless a large and specific degree of adjustability is required from the oscillator. This may be important if an electrically tunable oscillator is desired. In this case, a specific ratio of C_1 to C_0 could

5

© 1997 Microchip Technology Inc.

be specified. This would not be a low-cost item. It is customary to leave this blank if a specific value is not desired, or the words "as required" can be placed in that location.

Other alternative methods of specifying C_1 and C_0 might include a fractional frequency deviation between series and load capacity operation. The drive level should indicate the highest drive level which the PICmicro designer estimates the crystal will experience in operation. The crystal designer would like this value to be very low, rendering it a nonissue. The PICmicro designer must specify a practical maximum value and take steps to insure that it is not exceeded. Operation at spurious modes and activity dips are just two of the possible consequences of excessive drive levels. Activity dips are not to be tolerated if reliable operation is expected. To quantify this, a maximum allowable value is placed on the frequency deviation. This should be less than 1 PPM and must be less than 5 PPM at reasonable drive levels (i.e., less than 1 mW). The maximum permissible drive level is determined somewhat by the size of the blank, the electrode, and therefore the size of the holder chosen. The crystal manufacturer should offer a realistic value.

Temperature Characteristics

The crystal manufacturer must know the temperature range over which operation is intended, and this is the first item under the heading of temperature characteristics. The temperature profile of an A-T cut crystal is controlled by the angle of cut. The desired profile is also chosen in terms of angle. When purchasing a crystal however, do not attempt to specify an angle. These angles are referenced to the crystal's atomic lattice, and are calibrated using an X-ray diffraction technique. There is little direct correlation between manufacturers. Once again the measured results, in the form of measured temperature profiles, are much more accurate, and are the final word in any process control. The PICmicro designer should specify a fractional frequency deviation with tolerances between the turning points of the temperature profile. This is the accepted industry standard for specifying temperature performance, and any crystal manufacturer will readily accept it if the tolerances are realistic. A typical temperature profile might read; "turn-to-turn 5.5 PPM, + 5.0, - 3.5".

Packages

The type of package and the method of sealing it should be specified, although it may be useful on occasion to leave this item blank. Some manufacturers are equipped especially well for a particular type of holder or sealing process, and may offer a better package than required, at a competitive price. In general however, it is best to specify this at the outset. Package types (holders) greatly effect the price of a crystal. The primary performance effect is that of ageing, though other factors, such as thermal characteristics may also be effected.

Other Resonators

Specification of the NC-38 type crystal is limited to motional parameters, as it is not a "rotated" cut. There is little the crystal designer can do to alter the temperature profile. Specifying a ceramic resonator is mostly a matter of custom frequencies, but some control of motional parameters and package variations are possible, though not common. The large temperature profile tends to dominate all other considerations.

Crystal Example

The following is a specification for a 10 MHz A-T cut crystal which the PICmicro designer is likely to choose for a high stability application. The frequency tolerance is ± 20 PPM. A rather modest C_1 of .028 pF ± 20% is specified and the C_0, though not specified, will be around 5 to 7 pF. It is required to operate on frequency with a 32 pF load. The maximum drive level is 1 mW, and no activity dips greater than 3 PPM will be accepted. The crystal will operate over the temperature range of - 20°C to + 70°C. The frequency deviation between turnover points, is 5.3 + 4.5, - 3.2 PPM. Notice that the turning points do not necessarily fall within the operating temperature range (Figure 10). These deviations between turning points correspond to 1, 2 and 3 minutes of angle relative to the zero coefficient angle. Although the 1 minute curve displays a smaller deviation between the turn points, if it were the center of the angle range the lower end of 0 minutes would be unacceptable due to the rapid changes at the ends of the operating range, where cubed terms are in effect. The exact deviations were computer generated for each crystal angle. These offer more detail and accuracy than is possible with graphical techniques. An alternate method of specifying this is to set a total deviation over the entire operating temperature range of about ± 8 PPM. This is not as exact, and leaves the manufacturer more freedom to interpret requirements. If the PICmicro designer is not comfortable with these concepts, this may be the best approach. One may notice a small dissymmetry in the turning deviations. This is because the chosen operating temperature range is symmetrical at about 25°C, and the inflection temperature of the A-T cut is closer to 27°C or 28°C (Figure 10). In some cases the center of the operating temperature range may be very different from the inflection point of the crystal, and in order to realize the benefit of the best angle, a frequency offset at 25°C would be needed to center the temperature profile (this would not be part of the design sheet). The package is chosen to be an H/C 49 type which has a resistance weld seal. A maximum ageing rate of 2 PPM / year is required. No unusual shock or vibration is expected for this unit. Under the area of testing, temperature testing is required only on a sample of the lot. All units will be exposed to a thermal shock, and 10 days of ageing at 85°C. Ageing is of concern, so gross leakage is specified to be tested on all the units, and a fine leak test is to be performed on a 13% sample. Any notes about the application or special concerns would complete the crystal design sheet.

© 1997 Microchip Technology Inc.

THE PICMICRO ON BOARD OSCILLATOR(S)

PICmicro devices actually contains four complete oscillators which can be selected during the programming process. The selected oscillator is connected to the OSC1 and OSC2 pins, as well as the chip clock drivers by CMOS switches. In the windowed parts, these are all available to the programmer, while the OTP and QTP parts are pre-configured at the factory, and must be ordered as the desired type. The four types of oscillator available in the PIC16C5X/16CXXX series are:

RC (resistor capacitor)
LP (low power)
XT (crystal < 4 MHz)
HS (High speed)

The four types of oscillator available in the PIC17CXXX series are:

RC (resistor capacitor)
LF (low power)
XT (crystal < 4 MHz)
EC (External Clock)

The four circuits are shown in Figure 8. This unique arrangement gives the designer the ability to optimize the performance of the PICmicro in terms of clock speed, type of resonator, and power consumption.

The RC Oscillator

The RC oscillator is a relaxation type similar to the popular 555 timer. The OSC1 pin is the input to a Schmitt Trigger.

The LP oscillator

The LP, or low power oscillator, is designed to trade speed for low power operation. Although this circuit shares the same topology (schematic) as the XT oscillator, the transistors used in the LP oscillator have a higher Rdss value and draw considerably less current. This configuration is optimum for low frequency operation, because it trades the away unnecessary high frequency responses for dramatically reduced operating currents.

The XT oscillator

The XT oscillator is designed to give a compromise between high frequency performance and modest power consumption. The gain of this oscillator is as much as 15 times higher than the LP oscillator. This middle range will be used for frequencies up to 4 MHz.

The HS oscillator

The HS oscillator is designed to give the maximum gain and frequency response. The current consumption is accordingly higher. The gain is roughly five times higher than that of the XT oscillator. This gives the PICmicro the ability to operate at frequencies up to 20 MHz.

© 1997 Microchip Technology Inc.

FIGURE 13:CRYSTAL DESIGN SHEET

XYZ INC. **CRYSTAL DESIGN and TEST**

DRAWN _____ DATE_____

APPROVED _____DATE_____

FREQUENCY @ LOAD _____

MAKE TOLERANCE _____

MODE OF VIBRATION_____

SERIES RESISTANCE_____

MOTIONAL CAPACITANCE_____

STATIC CAPACITANCE _____

DRIVE LEVEL _____

SPURIOUS _____

ACTIVITY DIPS _____

AGEING _____

OPERATING TEMPERATURE RANGE _____

Frequency Deviation _____

_ From Turn to Turn

_ Over Operating Temperature range

PACKAGE_____

TYPE OF SEAL _____

EVIRONMENTAL:

VIBRATION_____

SHOCK_____

CRYSTAL TEST

TEMPERATURE_____

THERMAL SHOCK _____

AGEING _____

GROSS LEAK_____

FINE LEAK _____

NOTES

© 1997 Microchip Technology Inc.

FIGURE 14:EXAMPLE CRYSTAL DESIGN SHEET

XYZ INC. **CRYSTAL DESIGN and TEST**

DRAWN _____ DATE_____

APPROVED _____DATE_____

FREQUENCY @ LOAD 10 MHz @ 32 pF

MAKE TOLERANCE ± 20 PPM

MODE OF VIBRATION Fundamental

SERIES RESISTANCE 25 Ω Max.

MOTIONAL CAPACITANCE .028 pF ± 20%

STATIC CAPACITANCE _____

DRIVE LEVEL 1 mW Max.

SPURIOUS < - 3 dB

ACTIVITY DIPS < 3 PPM

AGEING < 2 PPM / Year

OPERATING TEMPERATURE RANGE 20 to + 70°C

Frequency Deviation _____

√ From Turn to Turn

_ Over Operating Temperature range

PACKAGE H/C 49

TYPE OF SEAL Resistance

EVIRONMENTAL: N/A

VIBRATION_____

SHOCK_____

CRYSTAL TEST

TEMPERATURE_____

THERMAL SHOCK _____

AGEING 10 Days at 85°C

GROSS LEAK 100%

FINE LEAK 13% AQL

NOTES

5

© 1997 Microchip Technology Inc.

APPENDIX A

The curves of Figure 10 are calculated using a general form developed by Bechmand in 1955. For any temperature *T*, a factional deviation from the frequency at the reference temperature T_0, is given in the form:

$$\frac{\Delta F}{F} = a(T - T_0) + b(T - T_0) + (T - T_0)^3$$

where:

a = $-5.15 \times 10^{-6} * °(\theta - \theta_0)$

b = $0.39 \times 10^{-9} - 4.7 \times 10^{-9} * °(\theta - \theta_0)$

c = $109.5 \times 10^{-12} - 2 \times 10^{-12} * °(\theta - \theta_0)$

$(\theta - \theta_0)$ = the difference between the intended angle and the zero temperature coefficient angle, in degrees of arc.

T0 is the reference temperature and is usually taken as 25°C. The zero temperature coefficient angle is approximately - 35.25° relative to the Y-axis. The exact angle which produces a zero temperature coefficient and the exact inflection temperature are both strongly dependent on several factors, including overtone and resonator geometry. A degree as a unit of angle is too coarse for sufficient resolution. The following coefficients are divided by 60 for units of minutes of arc.

REFERENCES AND BIBLIOGRAPHY (HIGHLY RECOMMENDED READING)

Ferking, M.E., *Crystal Oscillator Design and Temperature Compensation*, Van Norstrand Reinhold, New York, 1978.

Parzen, B., *Design of Crystal and other Harmonic Oscillators*, John Wiley and Sons, New York, 1983.

Salt, D., *The HY-Q Handbook of Quartz Crystal Devices*, Van Norstrand Reinhold, CO. Ltd, Berkshire, England.

This application note has been developed by:

Kim Peck
29 Wachusett Road
Needham, MA 02192
(617) 444-7748

© 1997 Microchip Technology Inc.

MICROCHIP

AN606

Low Power Design Using PICmicro™ Microcontrollers

Author: *Rodger Richey*
Microchip Technology Inc.

INTRODUCTION

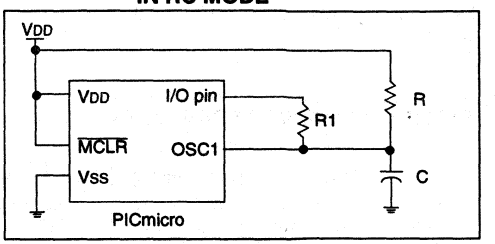
Power consumption is an important element in designing a system, particularly in today's battery powered world. The PICmicro family of devices has been designed to give the user a low-cost, low-power, and high-performance solution to this problem. For the application to operate at the lowest possible power, the designer must ensure that the PICmicro devices are properly configured. This application note describes some design techniques to lower current consumption, some battery design considerations, and suggestions to assist the designer in resolving power consumption problems.

DESIGN TECHNIQUES

Many techniques are used to reduce power consumption in the PICmicro devices. The most commonly used methods are SLEEP Mode and external events. These modes are the best way to reduce IPD in a system. The PICmicro device can periodically wake-up from Sleep using the Watchdog Timer or external interrupt, execute code and then go back into SLEEP Mode. In SLEEP Mode the oscillator is shut off, which causes the PICmicro device to consume very little current. Typical IPD current in most PICmicro devices is on the order of a few microamps.

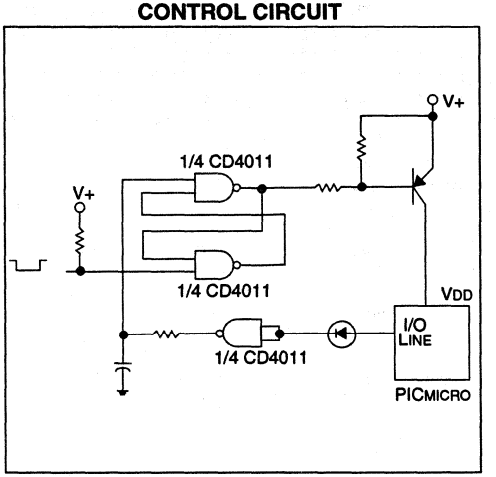
In cases where the PICmicro uses an RC oscillator but cannot use SLEEP Mode, another technique is used to lower power consumption. An I/O pin can remove a parallel resistance from the oscillator resistor while waiting for an event to occur. This would slow down the internal clock frequency, by increasing the resistance, and thus reduce Ipd. Once an event occurs the resistor can be switched in and the PICmicro device can process the event at full speed. Figure 1 shows how to implement this technique. The resistor **R1** would be used to increase the clock frequency by making the I/O pin an output and setting it to VDD.

FIGURE 1: USING AN EXTERNAL RESISTOR TO LOWER POWER IN RC MODE



External events can be used to control the power to PICmicro devices. For these cases, the Watchdog Timer can be disabled to further reduce current consumption. Figure 2 shows an example circuit that uses an external event to latch power on for the PICmicro device. Once the device has finished executing code, it disables power by resetting the latch. The latching circuit uses a low-power 4000 series CMOS quad chip which consumes a typical of 10 µA of current. The measured value of current consumption for the complete circuit with the PICmicro powered-down was 1 nA. Current consumption for a PICmicro in SLEEP Mode is typically 1 µA.

FIGURE 2: EXTERNAL EVENT POWER CONTROL CIRCUIT



5

© 1997 Microchip Technology Inc.

Power consumption is dependent on the oscillator frequency of the system. The device must operate fast enough to interface with external circuitry, yet slow enough to conserve power. The designer must account for oscillator start-up time, external circuitry initialization, and code execution time when calculating device power consumption. Table 1 shows various frequency oscillators, oscillator modes and the average current consumption of each mode. A PIC16C54 was used to collect data for Table 1 and the code is shown in Example 1. A current profile for a PIC16C54 in RC oscillator mode running at 261 kHz is shown in Figure 3. Figure 4 shows a current profile for a PIC16C54 in XT mode running at 1 MHz. The current profile includes three regions: power-up, active, and sleep. The power-up region is defined as the time the PICmicro device is in Power-on Reset and/or Oscillator Start-up Time. The active region is the time that the PICmicro device is executing code and the sleep region is the time the device is in SLEEP Mode. When using a 32.768 kHz crystal in LP oscillator mode, the designer must check that the oscillator has stabilized during the Power-on Reset. Otherwise, the device may not come out of reset properly.

TABLE 1: OSCILLATOR MODES

Osc. Type	Frequency	Osc. Mode	Power-up Region Current, Time	Active Region Current, Time	Sleep Region Current, Time
Resistor / Capacitor	261 kHz	RC	51.2 µA, 17.5 ms	396 µA , 12.8 ms	0.32 µA, 140 ms
Resistor / Capacitor	1.13 MHz	RC	61.4 µA, 17.5 ms	810 µA, 2.5 ms	0.3 µA, 140 ms
Crystal	32.768 kHz	LP	51.2 µA, 19 ms	23.5 µA, 93 ms	0.3 µA, 140 ms
Crystal	50 kHz	LP	61.4 µA, 16 ms	39.4 µA, 48.5 ms	0.28 µA, 140 ms
Crystal	1 MHz	XT	92 µA, 17.5 ms	443 µA, 3 ms	0.35 µA, 140 ms
Crystal	8 MHz	HS	123 µA, 18 ms	2.11 mA, 250 µs	0.3 µA, 140 ms
Resonator	455 kHz	XT	38.4 µA, 17.3 ms	421 µA, 7 ms	0.34 µA, 140 ms
Resonator	8 MHz	HS	143 µA, 18 ms	2.5 mA, 250 µs	0.29 µA, 140 ms

EXAMPLE 1: CURRENT PROFILE CODE

```
    TITLE "Current Profiling Program"
    LIST P=16C54, F=INHX8M
    INCLUDE "C:\PICMASTR\P16C5X.INC"
;*****************************************************************************
;*****************************************************************************
;;       This program initializes the PIC16C54, delays for 256 counts, then goes
;       to sleep. The WDT wakes up the PIC16C54.
;;*****************************************************************************
;*****************************************************************************
;Define General Purpose register locations
        LSB             EQU 0x10    ;delay control register
        Reset Vector
        ORG 0
START
        MOVLW           0x0B        ;WDT Prescaler of 1:8
        OPTION
        CLRF            PORTA       ;clear PORTA
        CLRF            PORTB       ;clear PORTB
        CLRW                        ;make PORTA and PORTB pins outputs
        TRIS            PORTA
        TRIS            PORTB
        CLRF            LSB
LOOP    DECFSZ          LSB,1
        GOTO            LOOP
        SLEEP                       ;go to sleep
        END
```


© 1997 Microchip Technology Inc.

FIGURE 3: CURRENT PROFILE (261 kHz RC MODE)

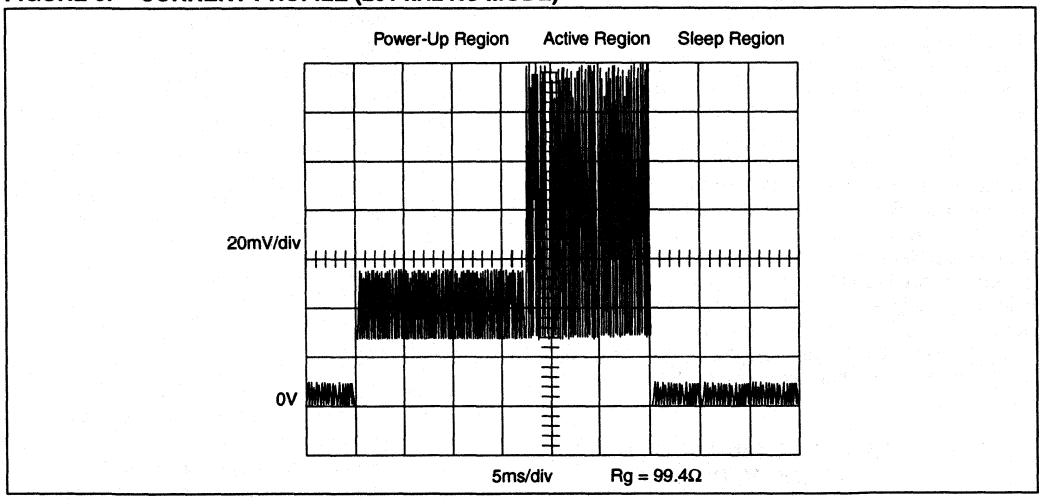
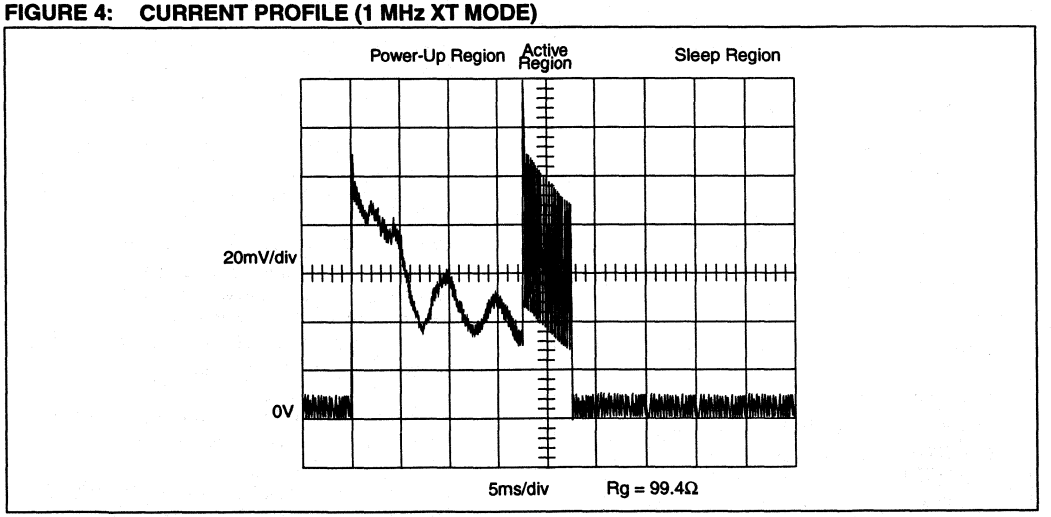


FIGURE 4: CURRENT PROFILE (1 MHz XT MODE)



Designing a system for lower supply voltages, typically 3V, is another method to reduce IPD. This type of design is best utilized in a battery powered system where current consumption is very low. A wide range of devices from op-amps and Analog-to-Digital (A/D) converters to CMOS logic products are being manufactured for low voltage operation. This gives the designer the flexibility to design a low voltage system with the same type of components that are available for a 5V design. Refer to the PICmicro device data sheets for IPD vs. VDD data.

Since any I/O pin can source or sink up to 20 mA, the PICmicro devices can provide power to other components. Simply connect the VDD pin of an external component to an I/O pin. Currently, most of the op-amps, A/D converters, and other devices manufactured today are low-power and can be powered by this technique. This provides the ability to turn off power to sections of the system during periods of inactivity.

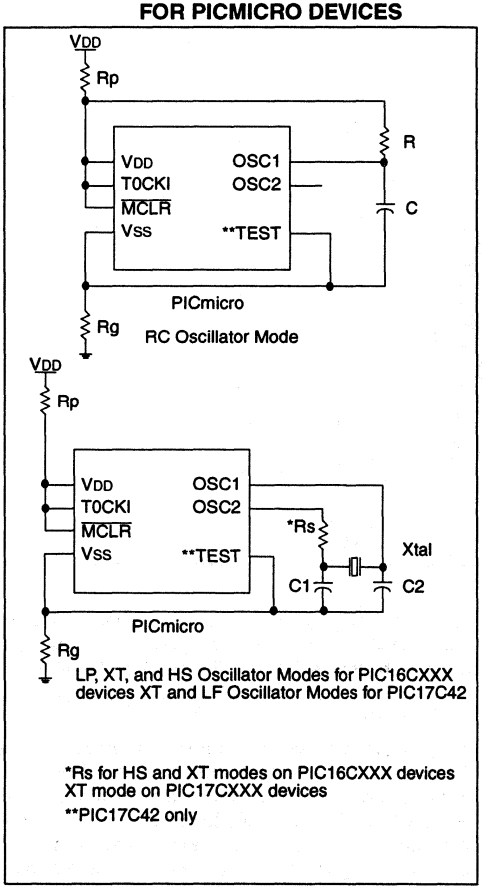
Temperature will effect the current consumption of the PICmicro devices in different ways. Typically devices will consume more current at extreme temperatures and batteries will have less available current at those same temperatures. PICmicro devices will exhibit higher IPD currents at high temperatures. Refer to the PICmicro device data sheets for IPD vs. Temperature data.

© 1997 Microchip Technology Inc.

TROUBLESHOOTING IPD

The first step in troubleshooting IPD problems is to measure the IPD that the circuit is consuming. Circuits to measure IPD for all oscillator modes are shown in Figure 5 for PICmicro devices. The resistor Rp is used to measure the amount of current entering the VDD pin when resistor Rg is shorted. The resistor Rg is used to measure the amount of current leaving the VSS pin when resistor Rp is shorted. The value of Rp and Rg should be approximately 100Ω for all oscillator modes. The two values of current should be approximately the same when the PICmicro is operating at the lowest possible power. If you find that the values of IPD measured from both configurations are not equivalent or are higher than the specifications, the following suggestions should help to find the source of extra current.

FIGURE 5: CIRCUITS TO MEASURE IPD FOR PICMICRO DEVICES



Basically, if Ip is not equal to Ig, then an I/O pin is either sourcing (IP>IG) current or sinking (IP<IG) current.

- Is the $\overline{MCLR}$ pin tied to VDD? Is the rate of rise of VDD slower than 0.05 V/ms? Does VDD start at VSS then rise? These conditions will not guarantee that the chip will come out of reset and function properly. Some of the circuits on PICmicro devices will start operating at lower voltage levels than other circuits. See Application Note AN522 "Power-Up Considerations" in the Microchip *Embedded Control Handbook*.
- Are all inputs being driven to VSS or VDD? If any input is not driven to either VSS or VDD, it will cause switching currents in the digital (i.e., flashing) input buffers. The exceptions are the oscillator pins and any pin configured as an analog input. During Power-on Reset or Oscillator Start-up time, pins that are floating may cause increased current consumption.
- All unused I/O pins should be configured as outputs and set high or low. This ensures that switching currents will not occur due to a floating input.
- Is the TMR0 (T0CKI) pin pulled to VSS or VDD? The TMR0 pin of PIC16C5X devices should be tied to VSS or VDD for the lowest possible current consumption.
- If an analog voltage is present at a pin, is that pin configured as an analog input? If an analog voltage is present at a pin configured as a digital input, the digital input buffers devices will consume more current due to switching currents.
- Are all on-chip peripherals turned off? Any on-chip peripheral that can operate with an external clock source, such as the A/D converter or asynchronous timers, will consume extra current.
- Are you using the PORTB internal pull-up resistors? If so and if any PORTB I/O pin is driving or receiving a zero, the additional current from these resistors must be considered in the overall current consumption.
- Is the Power-Up Timer being used? This will add additional current drain during power-up.
- If the currents measured at the Rp and Rg resistors are not the same, then current is being sourced or sunk by an I/O pin. Make sure that all I/O pins that are driving external circuitry are switched to a low power state. For instance, an I/O pin that is driving an LED should be switched to a state where the LED is off.
- Is the window of a JW package device covered? Light will affect the current consumption of a JW package device with the window left uncovered.

© 1997 Microchip Technology Inc.

IPD Analysis Using A Random Sample

The Microchip *1994 Microchip Data Book* specifies the typical IPD current for a PIC16C5X part at 4 µA and the maximum IPD current at 12 µA. These values are valid at a VDD voltage of 3V and a temperature range of 0°C to 70°C with the Watchdog Timer enabled. A control group of fifty PIC16C54's were randomly selected with pre-production and production samples. IPD tests were run on the group for a voltage range of 2.5V to 6.5V and for a temperature range of 0°C to 70°C. Table 2 compares the median and maximum values obtained by the IPD tests to the typical and maximum values in the data book. The IPD test data and the data book values are based on VDD = 3.0V, Watchdog Timer Enabled, and a temperature range of 0°C to 70°C.

The values in the data book are obtained from devices in which the manufacturing process has been skewed to various extremes. This should produce devices which function close to the minimum and maximum operating ranges for each parameter shown in the data book. The typical values obtained in the data book are actually the mean value of characterization data at a temperature of 25°C. The minimum and maximum values shown in the data book are the mean value of the characterization data at the worst case temperature, plus or minus three times the standard deviation. Statistically this means that 99.5% of all devices will operate at or below the typical value and much less than the maximum value.

TABLE 2: IPD COMPARISON OF CONTROL GROUP vs. DATA BOOK VALUES

Source	Typical or Median IPD	Maximum
Control Group	2.349 µA	3.048 µA
1994 Microchip Data Book	4 µA	12 µA

BATTERY DESIGN

When designing a system to use batteries, the designer must consider the maximum current consumption, operating voltage range, size and weight constraints, operating temperature range, and the frequency of operation. Once the system design is finished, the designer must again ask some questions that will define what type of battery to use. What is the operating voltage range? What is the current drain rate? What are the size constraints? How long will the system be used? What type of battery costs can be tolerated? What range of temperatures will the system be operated?

It is difficult to state a rule of thumb for selecting batteries because there are many variables to consider. For example, operating voltages vary from one battery type to another. Lithium cells typically provide 3.0V while Nickel-Cadmium cells provide 1.2V. On the other hand, Lithium cells can withstand minimal discharge rates while Nickel-Cadmium can provide up to 30A of current. A designer must consider all characteristics of each battery type when making a selection. Appendix B contains a simple explanation of batteries, a characteristic table for some common battery types, and discharge curves for the common batteries.

It is very important when doing a low power design to correctly estimate the required capacity of the power source. At this point, the designer should be able to estimate the operating voltage, current drain rates and how long the system is supposed to operate. To explain how to estimate the required capacity of a system, we will use the first entry from Table 1 using an RC oscillator set at 261 kHz. Figure 3 shows the current profile for this entry. It can be seen that the profile has a period of 170.3 ms with a 17.5 ms power-up region, a 12.8 ms active region, and a 140 ms sleep region. Assuming that the system will be required to operate for six months, we can now calculate the capacity required to power this system. Example 2 will illustrate the procedure. If a system does not have a periodic current profile, then the percentages obtained in step 1 of Example 2 will have to be estimated.

© 1997 Microchip Technology Inc.

EXAMPLE 2: CAPACITY CALCULATION

1. Calculate the percentage of time spent in power-up, active, and sleep regions.

   power-up
   (17.5 ms / 170.3 ms) x 100 = 10.3%

   active
   (12.8 ms / 170.3 ms) x 100 = 7.5%

   sleep
   (140 ms / 170.3 ms) x 100 = 82.2%

2. Calculate the number of hours in 6 months.

   6 months
   x (30 days / month)
   x (24 hours / day) = 4320 hours

3. Using the number of hours, percentages, and currents calculate the capacity for each period of time

   power-up
   4320 hours x 10.3% x 51.2 µA = 22.8 mAh

   active
   4320 hours x 7.5% x 396 µA = 128.3 mAh

   sleep
   4320 hours x 82.2% x 0.32 µA = 1.14 mAh

4. Sum the capacities of each period

   22.8 mAh + 128.3 mAh + 1.14 mAh = 152.2 mAh

The capacity required to operate the circuit for six months is 152.2 mAh. Example 2 does not take into consideration temperature effects or leakage currents that are associated with batteries. The load resistance of a battery is affected by temperature which in turn changes the available voltage and current; however, the self discharge rate is higher.

EXAMPLE DESIGN

A PIC16C54 with an LP oscillator of 32.768 kHz is used in this design. A Linear Technology low-power 12-bit A/D converter samples a temperature sensor. This data is transmitted via an LED at 300 baud to a receiver. The A/D converter, op-amp, and temperature sensor are powered from an I/O pin on the PIC16C54. The Watchdog Timer is enabled to periodically wake the system up from Sleep and take a sample. Figure 6 shows the schematic for the example design and Appendix A contains the source code.

This circuit has two operating modes, active and sleep. There was not a distinct power-up region in this design. In the circuit with the peripheral chips powered directly from the battery, the example design consumed 8 mA of current in the active mode and 6.5 mA in SLEEP Mode. With the peripheral chips powered from an I/O pin of the PIC16C54, the example design consumed 4 mA of current in the active mode and 0.5 µA in SLEEP Mode. The advantage of using an I/O pin to provide power to peripherals can be seen in a calculation of the capacity required to operate the circuit for one month. Example 3 details the two capacity calculations.

EXAMPLE 3: CAPACITY CALCULATION FOR THE EXAMPLE DESIGN

1. Calculate the percentage of time spent in the active and SLEEP Modes.

   active - battery power
   (210 ms / 2.61 s) x 100 = 8%

   sleep - battery power
   (2.4 s / 2.61 s) x 100 = 92%

   active - I/O power
   (188 ms / 2.638 s) x 100 = 7.1%

   sleep - I/O power
   (2.45 s / 2.638 s) x 100 = 92.9%

2. Calculate the number of hours in 1 month.

   1 month
   x (30 days / month)
   x (24 hours / day)
   = 720 hours

3. Using the number of hours, percentages and currents calculate the capacity for each period of time.

   active - battery power
   720 hours x 8% x 8 mA = 461 mAh

   sleep - battery power
   720 hours x 92% x 6.5 mA = 4306 mAh

   active - I/O power
   720 hours x 7.1% x 4 mA = 205 mAh

   sleep - I/O power
   720 hours x 92.9% x 0.5 µA = 0.4 mAh

4. Sum the capacities of each period.

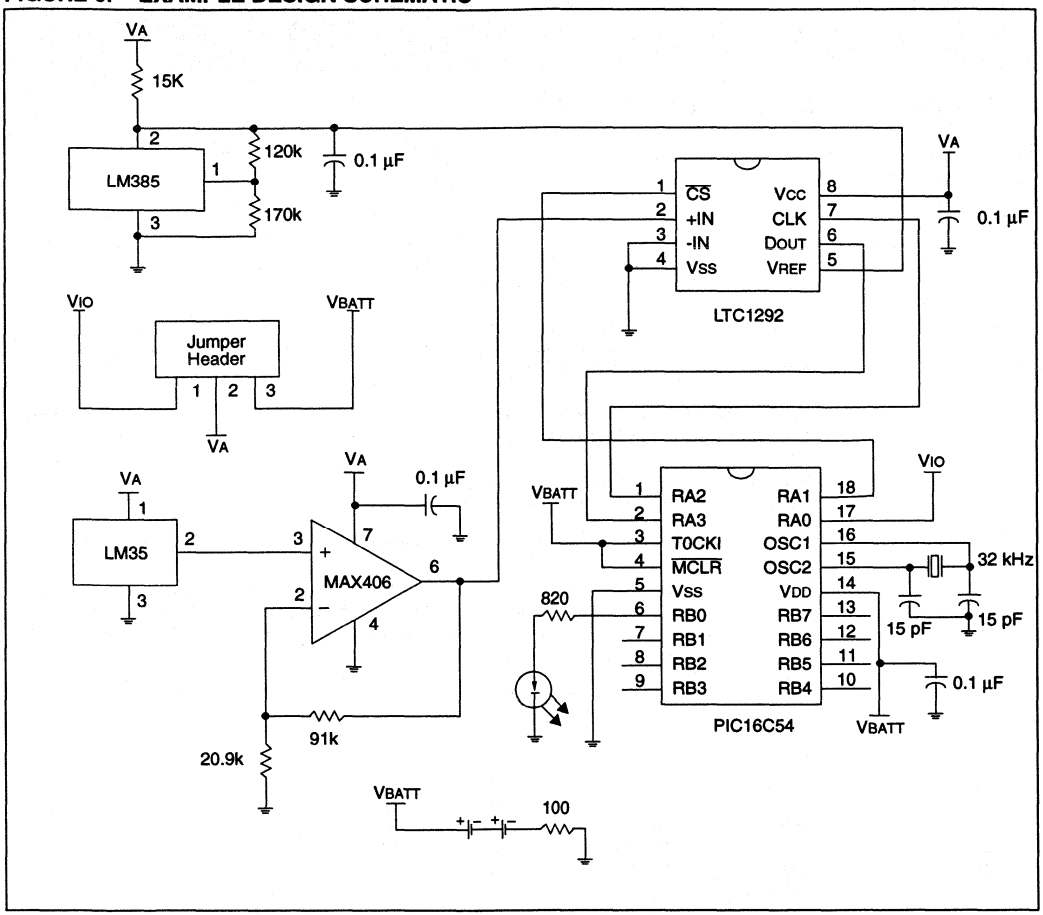
   battery power
   461 mAh + 4306 mAh = 4767 mAh

   I/O power
   205 mAh + 0.4 mAh = 206 mAh

The capacity required to operate this circuit for one month can be reduced by a factor of twenty just by powering the peripheral components from an I/O pin. The example design will use two Panasonic® BR2325 Lithium batteries in series to provide power to the circuit. This results in a Vbatt of 6V and a capacity of 165 mAh. Using the estimation process, the circuit should function for approximately 24 days. The actual time of operation was 24.2 days with the system running in an ambient temperature of 22°C.

© 1997 Microchip Technology Inc.

FIGURE 6: EXAMPLE DESIGN SCHEMATIC



SUMMARY

This application note has described some of the methods used to lower IPD and reduce overall system current consumption. Some obvious methods such as SLEEP Mode and low voltage design were given. Techniques such as powering components from I/O pins and oscillator mode and frequency selection can also be important in reducing IPD and overall system current. Some suggestions for troubleshooting IPD problems were presented. Finally, some considerations for designing a battery powered system were offered.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: EXAMPLE DESIGN CODE

```
MPASM 01.02.05 Released    LOWPWR.ASM   1-9-1995  13:2:42                PAGE  1
Ipd/Battery Apnote Example Design
LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     0001         TITLE "Ipd/Battery Apnote Example Design"
                     0002         LIST P=16C54, F=INHX8M
                     0003
                     0004         INCLUDE "P16C5X.INC"
                     0002 ;P16C5X.INC  Standard Header File, Ver. 0.1 Microchip Technology, Inc.
                     0004
                     0005
                     0006 ;*****************************************************************
                     0007 ;*****************************************************************
                     0008 ;
                     0009 ;       Filename:       lowpwr.asm
                     0010 ;       REVISION:       9 Jan 95
                     0011 ;
                     0012 ;*****************************************************************
                     0013 ;
                     0014 ;  This program initializes the PIC, takes a sample, and outputs the
                     0015 ;  value to PORTB pin 0 (the LED), and then goes to Sleep.  The
                     0016 ;  Watchdog Timer wakes the device up from Sleep.  PORTA pin 0 is used
                     0017 ;  to control power to peripherals.
                     0018 ;
                     0019 ;*****************************************************************
                     0020 ;*****************************************************************
                     0021
                     0022 ;       Define variable registers
  0010               0023         MSB             EQU     0x10
  0011               0024         LSB             EQU     0x11
  0012               0025         DELAY_CNT       EQU     0x12
  0013               0026         SHIFT           EQU     0x13
  0014               0027         COUNT           EQU     0x14
                     0028
                     0029 ;       Reset Vector
                     0030         ORG     0x1FF
  01FF 0A00          0031         GOTO    START
                     0032
                     0033 ;       Start of main code
                     0034         ORG     0
                     0035
                     0036 ;*****************************************************************
                     0037 ;       Main routine which initializes the device, and has main loop.
                     0038 ;*****************************************************************
0000                 0039 START
0000 0C2F            0040         MOVLW   0x2F            ;1:128 WDT PRESCALAR
0001 0002            0041         OPTION
0002 0C02            0042         MOVLW   0x02            ;RA1 SET HIGH
0003 0025            0043         MOVWF   PORTA
0004 0066            0044         CLRF    PORTB           ;ALL PINS SET TO Vss
0005 0C08            0045         MOVLW   0x08            ;RA3-DATA INPUT
0006 0005            0046         TRIS    PORTA           ;RA0-POWER,RA1-CS,RA2-CLOCK OUTPUTS
0007 0040            0047         CLRW                    ;PORTB ALL OUTPUTS, RB0-LED OUTPUT
0008 0006            0048         TRIS    PORTB
0009 0071            0049         CLRF    LSB             ;CLEAR A/D RESULT REGISTERS
000A 0070            0050         CLRF    MSB
                     0051
000B 0004            0052         CLRWDT
```


© 1997 Microchip Technology Inc.

```
000C 0911               0053          CALL    SAMPLE          ;GET SAMPLE FROM A/D
000D 0004               0054          CLRWDT
000E 0948               0055          CALL    OUTPUT          ;OUTPUT SAMPLE TO LED AT 300 BAUD
000F 0004               0056          CLRWDT
0010 0003               0057          SLEEP
                        0058
                        0059
                        0060 ;*************************************************************************
                        0061 ;        Main routine for retrieving a sample from the A/D.
                        0062 ;*************************************************************************
0011                    0063 SAMPLE
0011 0505               0064          BSF     PORTA,0         ;TURN POWER ON TO PERIPHERALS
0012 0943               0065          CALL    DELAY           ;WAIT FOR PERIPHERALS TO STABILIZE
0013 0C0B               0066          MOVLW   0x0B            ;DATA COUNTER, 12 BIT A/D
0014 0034               0067          MOVWF   COUNT
0015 0C08               0068          MOVLW   0x08            ;SET SHIFT REGISTER
0016 0033               0069          MOVWF   SHIFT
0017 0000               0070          NOP
0018 0425               0071          BCF     PORTA,1         ;ENABLE A/D
0019 0000               0072          NOP
001A 0545               0073          BSF     PORTA,2         ;1ST CLOCK RISE
001B 0000               0074          NOP
001C 0445               0075          BCF     PORTA,2         ;1ST CLOCK FALL
001D 0000               0076          NOP
001E 0545               0077          BSF     PORTA,2         ;NULL BIT CLOCK RISE
001F 0000               0078          NOP
0020 0445               0079          BCF     PORTA,2         ;NULL BIT CLOCK FALL
0021 0000               0080          NOP
                        0081
0022 0933               0082 LOOP     CALL    READ            ;READ DATA BIT
0023 0000               0083          NOP
0024 0545               0084          BSF     PORTA,2         ;BIT CLOCK RISE
0025 0000               0085          NOP
0026 0445               0086          BCF     PORTA,2         ;BIT CLOCK FALL
0027 0000               0087          NOP
0028 02F4               0088          DECFSZ  COUNT,F         ;CHECK LOOP COUNTER
0029 0A22               0089          GOTO    LOOP
002A 0933               0090          CALL    READ            ;READ LAST BIT
002B 0000               0091          NOP
002C 0545               0092          BSF     PORTA,2         ;SET CLOCK
002D 0000               0093          NOP
002E 0525               0094          BSF     PORTA,1         ;SET CS
002F 0000               0095          NOP
0030 0445               0096          BCF     PORTA,2         ;CLEAR CLOCK
0031 0405               0097          BCF     PORTA,0         ;POWER DOWN PERIPHERALS
0032 0800               0098          RETURN
                        0099
                        0100 ;*************************************************************************
                        0101 ;        Reads a bit from PORTA, data line from the A/D.
                        0102 ;*************************************************************************
0033                    0103 READ
0033 0004               0104          CLRWDT
0034 0774               0105          BTFSS   COUNT,3         ;CHECK IF AT BIT 8 - 11
0035 0A3B               0106          GOTO    RLOW            ;GOTO BITS 0 - 7
0036 0765               0107          BTFSS   PORTA,3         ;CHECK IF DATA IS CLEAR
0037 0A3F               0108          GOTO    REND            ;GOTO EXIT
0038 0213               0109          MOVF    SHIFT,W         ;ADD A ONE TO MSB IN THE CORRECT
0039 01F0               0110          ADDWF   MSB,F           ;BIT POSITION
003A 0A3F               0111          GOTO    REND
003B 0765               0112 RLOW     BTFSS   PORTA,3
003C 0A3F               0113          GOTO    REND
003D 0213               0114          MOVF    SHIFT,W         ;ADD A ONE TO LSB IN THE CORRECT
003E 01F1               0115          ADDWF   LSB,F           ;BIT POSITION
003F 0333               0116 REND     RRF     SHIFT,F         ;SHIFT
0040 0603               0117          BTFSC   STATUS,C        ;IF ONE IS IN THE CARRY
0041 0333               0118          RRF     SHIFT,F         ;SHIFT AGAIN
```


© 1997 Microchip Technology Inc.

```
0042 0800           0119         RETURN
                    0120
                    0121 ;*****************************************************************
                    0122 ;       Simple delay loop for 772 clock cycles.
                    0123 ;*****************************************************************
0043                0124 DELAY
0043 0004           0125         CLRWDT                  ;RESET WATCHDOG TIMER
0044 0072           0126         CLRF    DELAY_CNT
0045 02F2           0127 DLOOPL  DECFSZ  DELAY_CNT,F
0046 0A45           0128         GOTO    DLOOPL
0047 0800           0129         RETURN
                    0130
                    0131
                    0132 ;*****************************************************************
                    0133 ;       Output sample to LED at 300 baud.
                    0134 ;*****************************************************************
0048                0135 OUTPUT
0048 0C08           0136         MOVLW   0x08            ;SHIFT 8 MSB BITS OUT
0049 0034           0137         MOVWF   COUNT
                    0138
004A 0370           0139 MSBOUT  RLF     MSB,F          ;SHIFT LSB INTO CARRY
004B 0703           0140         BTFSS   STATUS,C       ;IF CARRY IS SET
004C 0A50           0141         GOTO    MSBCLR
004D 0506           0142         BSF     PORTB,0        ;SET PORTB,0
004E 0968           0143         CALL    BAUD
004F 0A54           0144         GOTO    MSBCHK         ;CHECK FOR ALL 8 BITS TO BE SENT
0050 0406           0145 MSBCLR  BCF     PORTB,0        ;OTHERWISE CLEAR PORTB,0
0051 0000           0146         NOP                    ;WAIT TO SET BAUD RATE 600
0052 0000           0147         NOP
0053 0968           0148         CALL    BAUD
0054 02F4           0149 MSBCHK  DECFSZ  COUNT          ;CHECK FOR ALL 8 BITS TO BE SENT
0055 0A4A           0150         GOTO    MSBOUT
                    0151
0056 0C08           0152         MOVLW   0x08            ;SHIFT 8 LSB BITS OUT
0057 0034           0153         MOVWF   COUNT
                    0154
0058 0371           0155 LSBOUT  RLF     LSB,F          ;SHIFT LSB INTO CARRY
0059 0703           0156         BTFSS   STATUS,C       ;IF CARRY IS SET
005A 0A5E           0157         GOTO    LSBCLR
005B 0506           0158         BSF     PORTB,0        ;SET PORTB,0
005C 0968           0159         CALL    BAUD
005D 0A62           0160         GOTO    LSBCHK         ;CHECK FOR 8 BITS TO BE SENT
005E 0406           0161 LSBCLR  BCF     PORTB,0        ;OTHERWISE CLEAR PORTB,0
005F 0000           0162         NOP                    ;WAIT TO SET BAUD RATE 600
0060 0000           0163         NOP
0061 0968           0164         CALL    BAUD
0062 02F4           0165 LSBCHK  DECFSZ  COUNT          ;CHECK FOR 8 BITS TO BE SENT
0063 0A58           0166         GOTO    LSBOUT
0064 0406           0167         BCF     PORTB,0        ;CLEAR PORTB,0
0065 0071           0168         CLRF    LSB            ;CLEAR LSB
0066 0070           0169         CLRF    MSB            ;CLEAR MSB
0067 0800           0170         RETURN
                    0171
                    0172 ;*****************************************************************
                    0173 ;       Delay loop for sending data to the LED at 300 baud.
                    0174 ;*****************************************************************
0068                0175 BAUD
0068 0000           0176         NOP
0069 0000           0177         NOP
006A 0000           0178         NOP
006B 0000           0179         NOP
006C 0000           0180         NOP
006D 0000           0181         NOP
006E 0000           0182         NOP
006F 0000           0183         NOP
0070 0000           0184         NOP
```


© 1997 Microchip Technology Inc.

```
0071 0000               0185          NOP
0072 0000               0186          NOP
0073 0000               0187          NOP
0074 0000               0188          NOP
0075 0800               0189          RETURN
                        0190
                        0191          END
                        0192
                        0193

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXX----------

0180 : ---------------- ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.


Errors   :     0
Warnings :     0
Messages :     0
```

5

APPENDIX B: BATTERY DESCRIPTIONS

Presently there are two types of batteries that are manufactured, primary and secondary. Primary batteries are those that must be thrown away once their energy has been expended. Low current drain, short duty cycles, and remote operation favor primary batteries such as Carbon Zinc and Alkaline. Secondary batteries can be recharged once they have exhausted their energy. High current drain or extended usage favors secondary batteries especially when the cost of replacement of disposable batteries is not feasible. Secondary batteries include Nickel-Cadmium and Nickel Metal Hydride.

A battery may be discharged by different means depending on the type of load. The type of load will have a significant effect on the life of the battery. The typical modes of discharge are constant resistance, constant current, and constant power. Constant resistance is when the load maintains a constant resistance throughout the discharge cycle. Constant current is the mode where the load draws the same current during discharge. Finally, constant power is defined as the current during a discharge increases as the battery voltage decreases.

The constant resistance mode results in the capacity of the battery being drained at a rapid and excessive rate, resulting in a short life. This is caused by the current during discharge following the drop in battery voltage. As a result, the levels of current and power during discharge are in excess of the minimum required.

The constant current mode has lower current and power throughout the discharge cycle than the constant resistance mode. The average current drain on the battery is lower and the discharge time to the end-voltage is longer.

The constant power discharge mode has the lowest average current drain and therefore has the longest life. During discharge, the current is lowest at the beginning of the cycle and increases as the battery voltage drops. Under this mode the battery can be discharged below its end voltage, because the current is increased as the voltage drops. The constant power mode provides the most uniform performance throughout the life of the battery and has the most efficient use of the energy in the battery.

The nominal voltage is the no-load voltage of the battery, the operating voltage is the battery voltage with a load, and the end-of-life voltage is the voltage when the battery has expended its energy. Energy Density is used to describe the amount of energy per unit of volume or mass (Wh/kg or Wh/l). Generally, energy density decreases with decreasing battery size within a particular type of battery. Most batteries are rated by an amp-hour (Ah) or milliamp-hour (mAh) rating. This rating is based on a unit of charge, not energy. A 1-amp current corresponds to the movement of 1 coulomb (C) of charge past a given point in 1 second (s). Table B-1 lists some typical characteristics of the most common types of batteries.

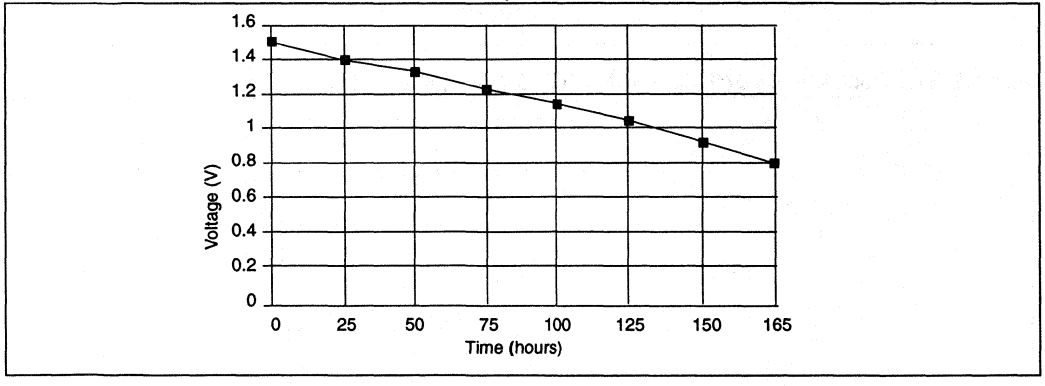
© 1997 Microchip Technology Inc.

TABLE B-1: TYPICAL BATTERY CHARACTERISTICS

	Carbon Zinc	Alkaline	Nickel Cadmium	Lithium	Nickel Metal Hydride	Zinc Air	Silver Oxide
Cell Voltage							
Nominal	1.5	1.5	1.2	3.0	1.2	1.4	1.6
Operating	1.25-1.15	1.25-1.15	1.25-1.00	2.5-3.0	1.25-1.0	1.35-1.1	1.5
End of life	0.8	0.9	0.9	1.75	0.9	0.9	0.9
Operating Temperature	-5°C to 45°C	-20°C to 55°C	-40°C to 70°C	-30°C to 70°C	-20°C to 50°C	0°C to 45°C	-20°C to 50°C
Energy Density (Wh/kg)	70	85	30	300	55	300	100
Capacity	60mAh to 18Ah	30mAh to 45Ah	150mAh to 4Ah	35mAh to 4Ah	500mAh to 5Ah	50mAh to 520mAh	15mAh to 210mAh
Advantages		High capacity, good low temp	good low temp, good high rate discharge	good low and high temp, good high rate discharge, long shelf life	better capacity than Nicad for same size	high energy density, good shelf life	good low temp, good shelf life
Limitations	Low energy density, poor low temp, poor high rate discharge		poor low rate discharge, disposal hazards	Violent reaction to water		Cannot stop reaction once started	poor high rate discharge
Relative Cost	low	low	medium	high	high	high	high
Type	**Primary**	**Primary**	**Secondary**	**Primary**	**Secondary**	**Primary**	**Primary**

Typical discharge curves for alkaline, carbon zinc, lithium, nickel cadmium, nickel metal hydride, silver oxide, and zinc air are shown in Figure B-1 through Figure B-7. These curves are only typical representations of each battery type and are not specific to any battery manufacturer. Also the load and current drain are different for each type of battery.

FIGURE B-1: ALKALINE DISCHARGE CURVE (16 mA LOAD)



© 1997 Microchip Technology Inc.

FIGURE B-2: CARBON ZINC DISCHARGE CURVE (16 mA LOAD)

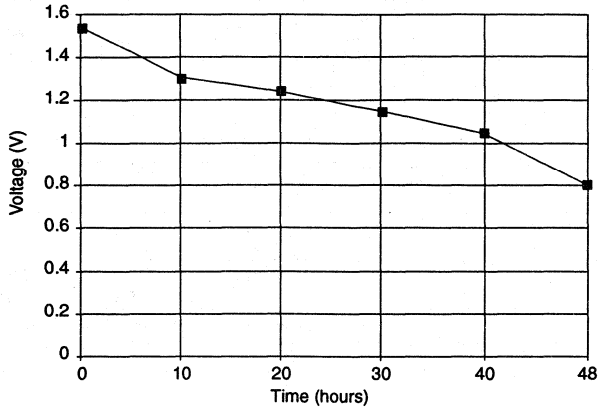


FIGURE B-3: LITHIUM DISCHARGE CURVE (2.8 mA LOAD)

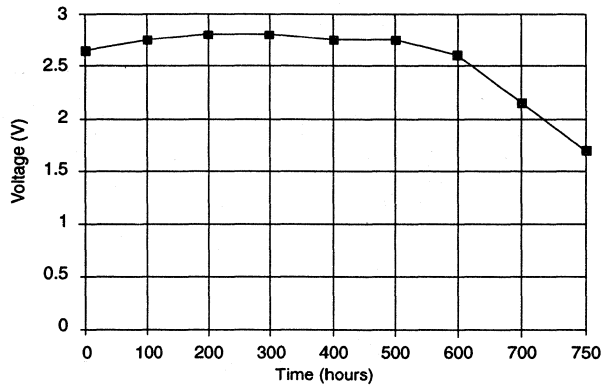
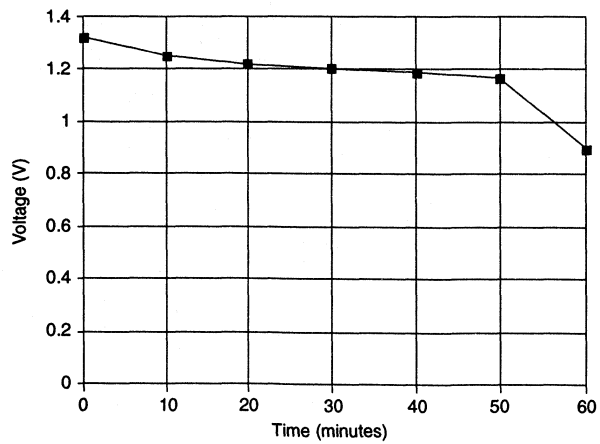


FIGURE B-4: NICKEL CADMIUM DISCHARGE CURVE (500 mA LOAD)



© 1997 Microchip Technology Inc.

FIGURE B-5: NICKEL METAL HYDRIDE DISCHARGE CURVE (1500 mA LOAD)

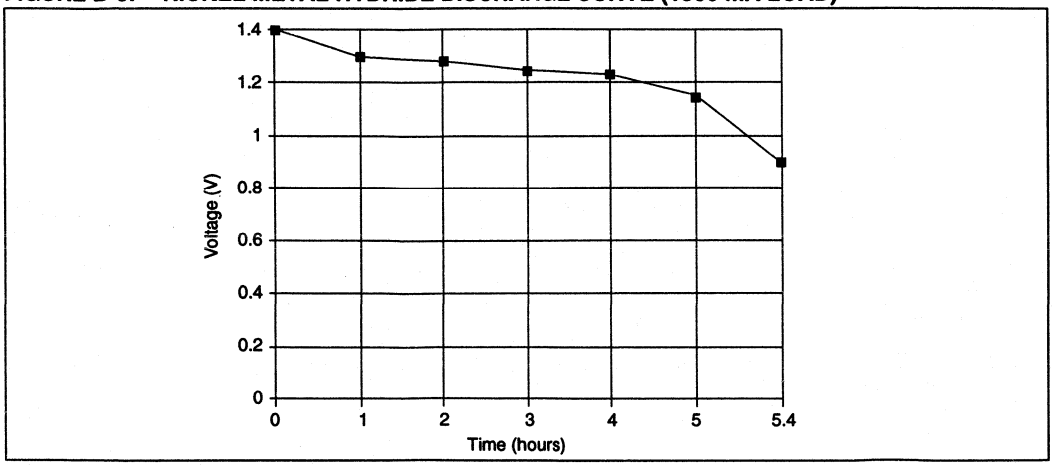


FIGURE B-6: SILVER OXIDE DISCHARGE CURVE (1 mA LOAD)

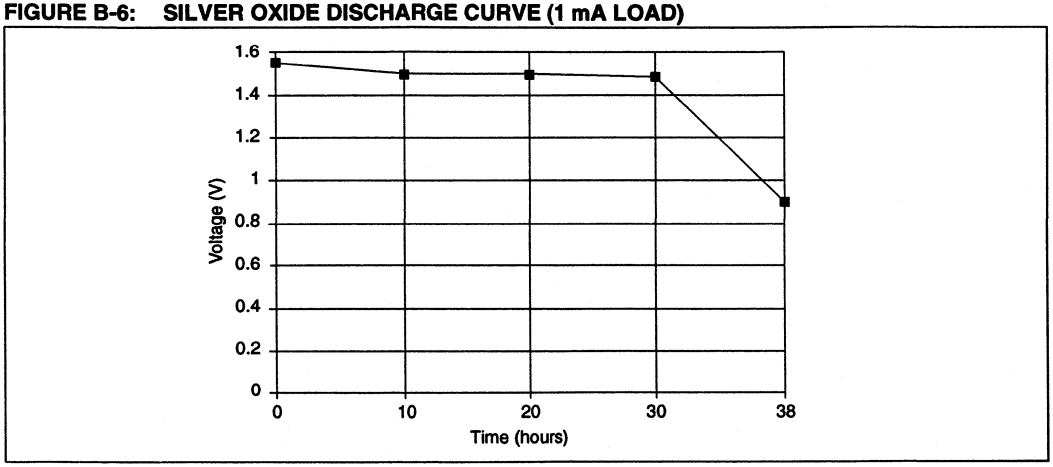
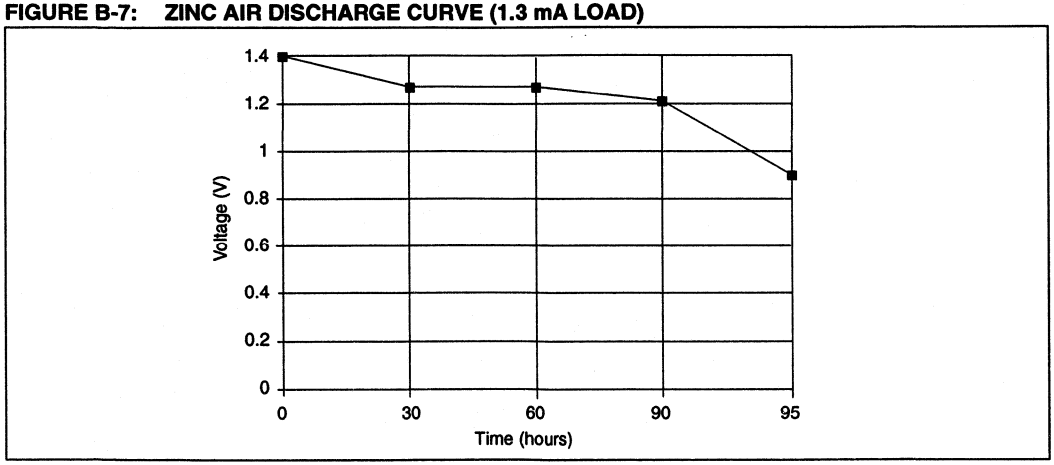


FIGURE B-7: ZINC AIR DISCHARGE CURVE (1.3 mA LOAD)



© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Power-up Trouble Shooting

Author: *Mark Palmer*
Microchip Technology Inc.
Contributions: *Richard Hull & Randy Yach*
Microchip Technology Inc.

INTRODUCTION

For any application to begin proper operation, the application must power-up properly. Many criteria must be taken into account to ensure this. The PICmicro™ devices integrate several features to simplify the design for the power-up sequence. These integrated features also reduce the total system cost.

This application note describes the requirements for the device to properly power-up, common pitfalls that designers encounter, and methods to assist in solving power-up problems.

THE POWER-UP SEQUENCE

There are several factors that determine the actual power-up sequence that a device will go through. These factors are:

- The Processor Family
  - PIC16C5X (Baseline)
  - PIC16CXXX (Midrange)
  - PIC17CXXX (High-end)
- Oscillator Configuration
- Device Configuration
- $\overline{MCLR}$ pin

Note: The PIC16CXXX family refers to devices with a 14-bit instruction word. This does not include the PIC16C5X family.

The Power-on Reset (POR) signal generation is discussed, followed by the power-up sequence for the specific device families.

Power-on Reset (POR) signal

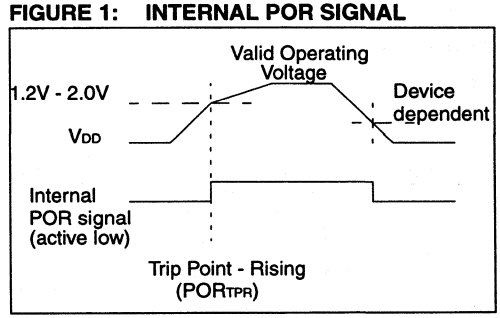
The data sheets show a Power-on Reset (POR) pulse, as in Figure 1. The POR signal is a level triggered signal. This representation may help in the understanding of future devices, which may have a brown-out reset capability.

The power-up sequence begins by increasing the voltage on the V_{DD} pin (from 0V). If the slope of the V_{DD} rise time is faster than 0.05 V/ms, the internal circuitry is capable of generating an internal reset signal. Depending on the device family, different power-up sequences will occur after this POR signal.

If the slope is less then 0.05 V/ms, the $\overline{MCLR}$ pin should be held low, by external circuitry, until a valid operating V_{DD} level is reached.

The V_{DD} rise time specification needs to be met, until the POR signal is generated. After the POR signal is generated the slope of the V_{DD} rise can change (to a faster or slower rise). This may have other ramifications, see the "Power-up Consideration" section. In general, the POR signal will trip (POR_{TP}) somewhere between 1.2V to 2.0V (Figure 1).

FIGURE 1: INTERNAL POR SIGNAL



When V_{DD} is falling, the voltage at which the internal POR signal returns to a low level is processor/device dependent. To ensure that a device will have a POR, the device voltage must return to V_{SS} before power is re-applied.

Note: Some devices (with EPROM program memory) have a newer POR circuit that does not require V_{DD} to return to V_{SS}. See the device data sheet for the complete specification on the POR operation.

The POR will be generated regardless of the level of the $\overline{MCLR}$ pin. The PICmicro device families are different on what triggers the power-up sequence. Table 1 describes the events that cause the POR sequence to occur.

After reaching the POR trip point (POR_{TPR}), the POR sequence holds the device in reset for a given time. Once this time has elapsed, the device voltage must be valid or the $\overline{MCLR}$ pin must be low. The time from the POR rising edge to the time that V_{DD} must be valid level is the $T_{POR2VDDV}$ time.

© 1997 Microchip Technology Inc.

TABLE 1: EVENTS THAT TRIGGER POR SEQUENCE

Device	Events
PIC16C5X	Both the POR signal rising edge and any $\overline{MCLR}$ rising edge(1)
PIC16CXXX	The POR signal rising edge
PIC17CXXX	Either the POR signal rising edge or the first $\overline{MCLR}$ rising edge (if $\overline{MCLR}$ is low when the POR occurs). After this event, all following $\overline{MCLR}$ rising edges(1) cause the device to start program execution immediately.

Note 1: The POR low-to-high transition onfigures Special Function Register (SFR) bits/registers to a specified value. The SFR bits/register are not identically affected by the $\overline{MCLR}$ signal. Refer to the device data sheet to see how the bits are affected by these two conditions.

The POR sequence for each of the PICmicro families is described in the following three sections:

PIC16C5X Family

PIC16CXXX Family

PIC17CXXX Family

PIC16C5X Family

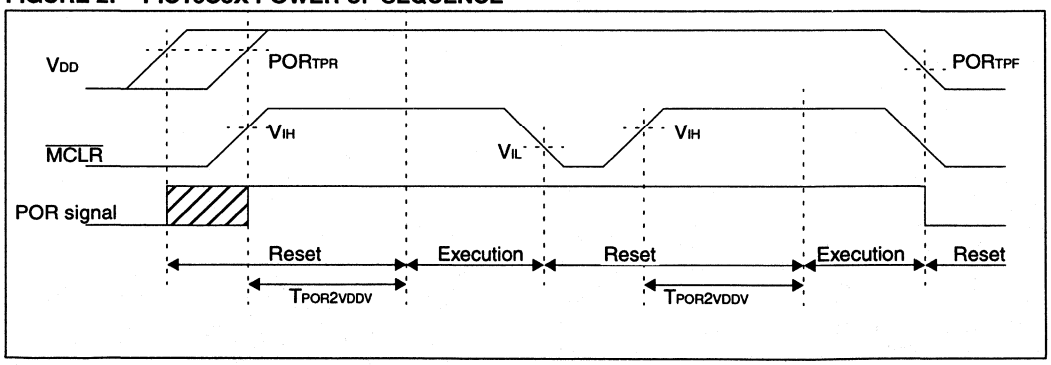
After the $\overline{MCLR}$ pin has reached a high level, the device is held in reset for typically 18 ms. This time is determined by an on-chip RC oscillator and 8-bit ripple counter. This Device Reset Timer (DRT), allows most crystals (except low frequency crystals) to start-up and stabilize. Due to the characteristics of resistors and capacitors, this time is extremely variable over temperature and voltage. There is also a device to device variation. See the data sheet for the range of this time-out.

TABLE 2: TIME-OUT IN VARIOUS SITUATIONS (TYPICAL)

Oscillator Configuration	Power-up	Wake-up from SLEEP
XT, HS, LP(1)	18 ms	18 ms
RC	18 ms	18 ms

Note 1: 32 kHz crystals have a typical start-up time of 1-2 seconds. Crystals >100 kHz have a typical start-up time of 10-20 ms. Resonators are typically <1 ms. All these times are voltage dependent.

FIGURE 2: PIC16C5X POWER-UP SEQUENCE



© 1997 Microchip Technology Inc.

PIC16CXXX Family

After the POR rising edge has occurred, the device can have up to 2 time-out sequences that occur in series. The first being the Power-up Timer (PWRT), the second being the Oscillator Start-up Timer (OST).

The Power-up Timer time-out will occur if enable bit PWRTE is read as a '1'. The PWRT uses a 10-bit counter, with the clock from an internal RC. Due to the characteristics of resistors and capacitors, this time is extremely variable over temperature and voltage. There is also a device to device variation. See the data sheet for the range of this time-out.

Note: Future devices will change the polarity of the PWRTE configuration bit. Refer to the specific data sheet for the polarity of this bit.

The OST will occur on power-up/wake-up when the device has oscillator mode selected. This allows the oscillator to stabilize before program execution begins. The OST uses a 10-bit counter, with the clock from the OSC pin. The time is dependent on the frequency of the input clock. This timer is disabled if the oscillator is configured as RC.

Figure 3 shows how the two timers work in the power-up sequence. V_{DD} must be valid when program execution starts. The T_{PWRT} + T_{OST} times can be thought of as the time that the device gives for the V_{DD} to become valid ($T_{POR2VDDV}$). Figure 4 shows when device execution begins for the case of the $\overline{MCLR}$ pin going high before $T_{POR2VDDV}$ times out. Figure 5 shows when the $\overline{MCLR}$ pin is held low longer than the $T_{POR2VDDV}$ time. The device starts execution immediately when $\overline{MCLR}$ goes high. Table 3 gives typical reset times.

TABLE 3: TIME-OUT IN VARIOUS SITUATIONS (TYPICAL)

Oscillator Configuration	Power-up		Wake-up from SLEEP
	PWRTE = 1 (2)	PWRTE = 0 (2)	
XT, HS, LP(1)	72 ms + 1024 Tosc	1024 Tosc	1024 Tosc
RC	72 ms	—	—

Note 1: 32 kHz crystals have a typical start-up time of 1-2 seconds. Crystals >100 kHz have a typical start-up time of 10-20 ms. Resonators are typically <1 ms. All these times are voltage dependent.

2: Future devices will change the polarity of this configuration bit. Refer to the specific data sheet for the polarity of the PWRT Configuration Bit.

FIGURE 3: PIC16CXX POWER-UP SEQUENCE

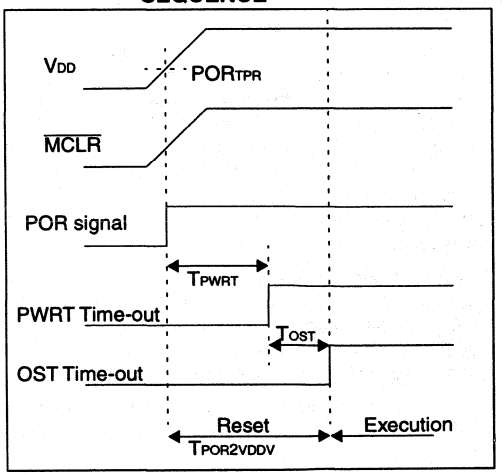


FIGURE 4: START OF DEVICE OPERATION ($\overline{MCLR}$ < $T_{POR2VDDV}$)

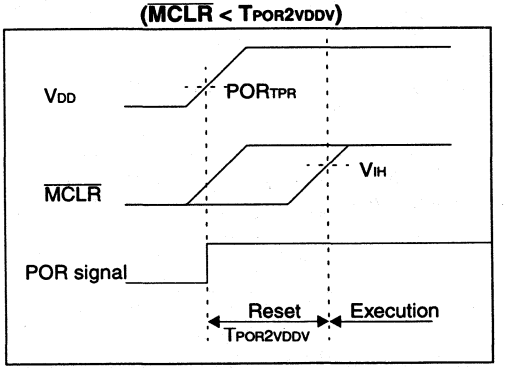
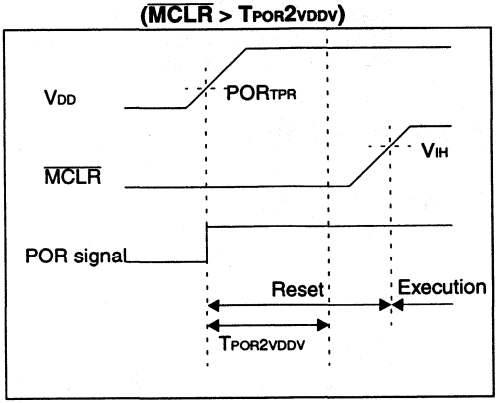


FIGURE 5: START OF DEVICE OPERATION ($\overline{MCLR}$ > $T_{POR2VDDV}$)



PIC17CXXX Family

When the $\overline{MCLR}$ pin comes to a high level, after the POR rising edge, the device has 2 time-out sequences that occur in parallel. One is the Power-up Timer (PWRT), the other is the Oscillator Start-up Timer (OST). The timer with the greater time holds the device in reset. Figure 6 shows the sequence with $\overline{MCLR}$ tied to V_{DD}. Figure 7 show the time-out when $\overline{MCLR}$ is independent of V_{DD}. The PWRT time is generally longer, except for low frequency crystals/resonators. The OST time does not include the start-up time of the oscillator/resonator.

The PWRT uses a 10-bit counter, with the clock from an internal RC. The characteristics of the RC vary from device to device and over temperature and voltage. The specification for the time-out range can be found in the electrical specification of the data sheet.

The OST uses a 10-bit counter, with the clock from the OSC pin. The time is dependent on the frequency of the input clock.

Until $\overline{MCLR}$ has reached a high level, the POR sequence will not start. While the POR signal remains high, all following $\overline{MCLR}$ pulses will not cause the POR sequences to occur (Figure 8).

TABLE 4: TIME-OUT IN VARIOUS SITUATIONS (TYPICAL)

Oscillator Configuration	Power-up	Wake-up from Sleep
RC, EC	Greater of 80 ms and 1024 Tosc	—
XT, LF(1)	Greater of 80 ms and 1024 Tosc	1024 Tosc

Note 1: 32 kHz crystals have a typical start-up time of 1-2 seconds. Crystals >100 kHz have a typical start-up time of 10-20 ms. Resonators are typically <1 ms. All these times are voltage dependent.

FIGURE 6: PIC17CXX POWER-UP SEQUENCE ($\overline{MCLR}$ TIED TO V_{DD})

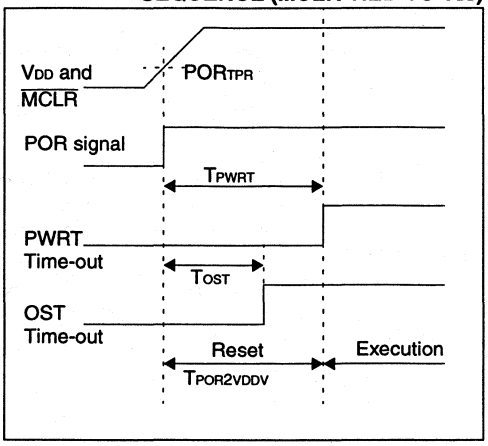


FIGURE 7: PIC17CXX POWER-UP SEQUENCE ($\overline{MCLR}$ NOT TIED TO V_{DD})

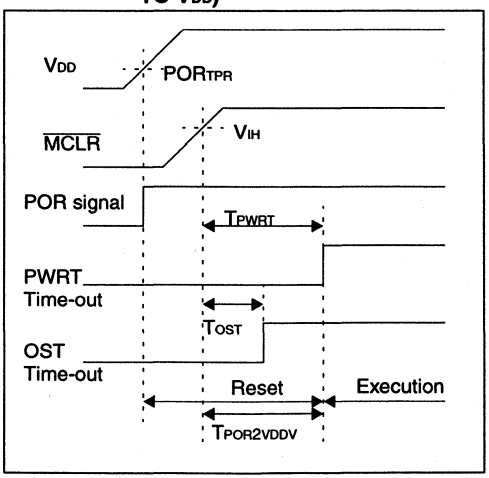
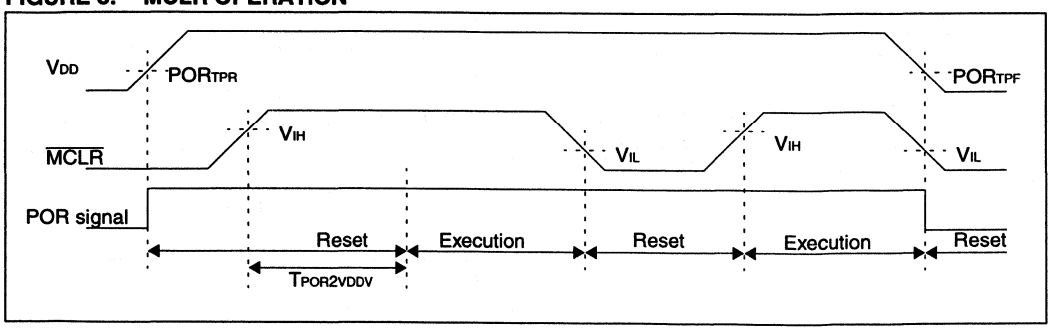


FIGURE 8: $\overline{MCLR}$ OPERATION



© 1997 Microchip Technology Inc.

POWER-UP CONSIDERATIONS

The device must be at a valid operating voltage when the device exits reset. This can be done by ensuring that the power supply rise time is fast enough to guarantee an operating V_{DD} level, or by using an external reset circuit which will hold $\overline{MCLR}$ low until the operating V_{DD} level is reached.

When the rise time of V_{DD} is very fast, there will be a time delay before the Power-on Reset (POR) signal will rise to a logic high ($T_{TP2PORH}$). This delay is in the 1-5 µs range, as shown in Figure 9.

Figure 10, Figure 11, and Figure 12 show the maximum time from the POR sequence beginning to the device having a valid operating voltage. Table 5 gives the $T_{POR2VDDV}$ times. When determining the time at which V_{DD} must be valid, the POR trip point must be assumed to be at the minimum POR voltage trip point.

How Crystal Frequencies affect Start-up time

Both the PIC16CXXX and PIC17CXXX families may have start-up times that include the contributions of the oscillator. Table 5 shows how the oscillator can affect each mode of operation, with Table 6 giving the reset time that an oscillator generates. This time can be used in the equation to calculate the total reset time, at the given frequency. This time may vary slightly due to the initial start-up characteristics of the crystal/oscillator circuit.

Note 1: The rise time specification does not ensure that a valid V_{DD} operating voltage will be reached before the device exits reset. The device's V_{DD} must be within the specified operating range for proper device operation.

Note 2: The start-up characteristics of the crystal/oscillator must also be taken into account when determining the time that the device must be held in reset.

TABLE 5: MAXIMUM TIME FROM POR RISING EDGE TO VALID V_{DD} VOLTAGE

	Osc Mode	Maximum Time	Conditions
PIC16C5X	LP, XT, HS, and RC	9 ms	
PIC16CXXX	RC	28 ms	
	LP, XT, and HS	28 ms + 1024 Tosc	PWRTE = 1
	LP, XT, and HS	1024 Tosc	PWRTE = 0
PIC17CXXX	LF, XT, EC, and RC	Greater of (40 ms or 1024 Tosc)	

FIGURE 9: POR DELAY FOR FAST V_{DD} RISE TIME

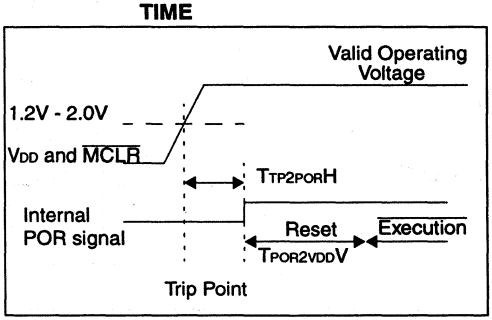


TABLE 6: RESET TIME DUE TO OSCILLATOR

	Clock Frequency								
	32 kHz	1 MHz	2 MHz	4 MHz	8 MHz	10 MHz	16 MHz	20 MHz	25 MHz
1024 Tosc	32 ms	1.0 ms	512 µs	256 µs	128 µs	102.4 µs	64 µs	51.2 µs	41 µs

FIGURE 10: MAXIMUM POWER-UP TIME, $\overline{MCLR}$ TIED TO V_{DD} (PIC16C5X, PIC16CXXX, PIC17CXXX)

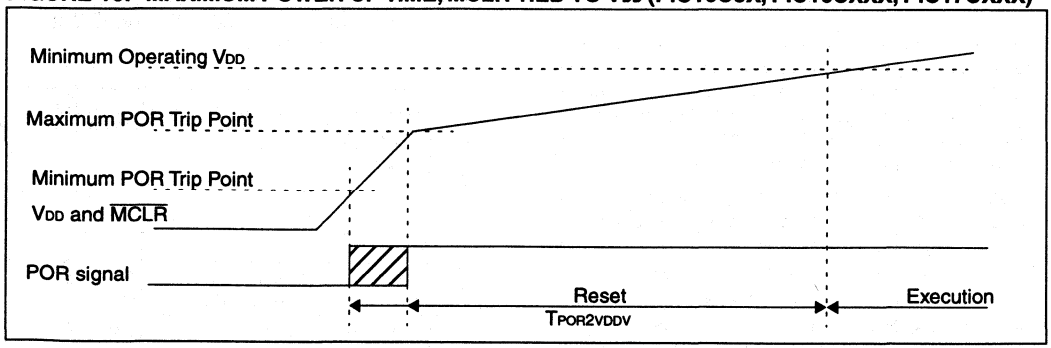


FIGURE 11: MAXIMUM POWER-UP TIME, $\overline{MCLR}$ NOT TIED TO V_{DD} (PIC16CXXX)

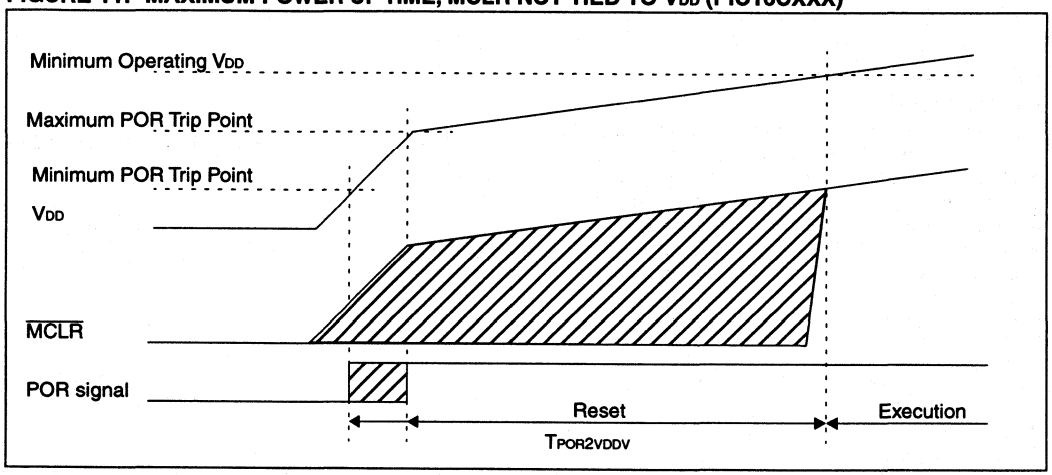
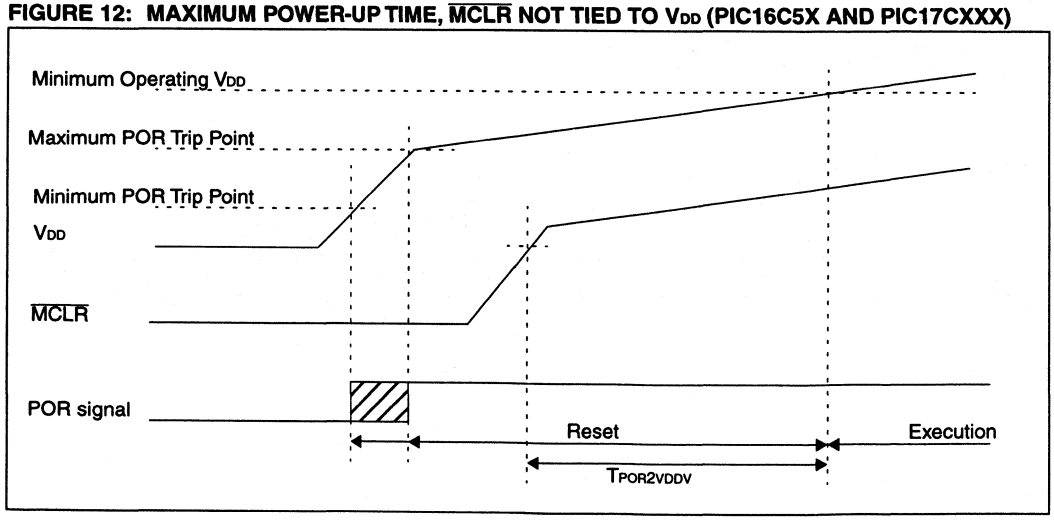


FIGURE 12: MAXIMUM POWER-UP TIME, $\overline{MCLR}$ NOT TIED TO V_{DD} (PIC16C5X AND PIC17CXXX)



© 1997 Microchip Technology Inc.

Oscillator and Resonator Considerations

Oscillators and resonators from different manufacturers may have different characteristics. The recommended capacitor selection can be found in each device's data sheet. When we do the capacitor selection, during the oscillator/resonator characterization, we are currently using devices from one of several manufacturers. Generally we use oscillators from either ECS, CTS, FOX or Epson, and ceramic resonators from either Murata Erie or Panasonic. Other manufacturers may be used in the future, depending on availability and other factors.

Other manufacturers devices may have significantly different characteristics. To ensure proper oscillator operation, the circuit should be verified at the lowest temperature/highest V_{DD} (to ensure that the crystal is not overdriven), and with the highest temperature/lowest V_{DD} (to ensure the device still starts up) that the device will be subjected to while in the application. This ensures a stable start-up and frequency for this device, at the extreme conditions of the application.

For production purposes, the above testing should be done with many different samples of the components selected. This is so the part to part variation of the capacitors, resistors, crystals/resonators, and PICmicro devices are taken into account. All PICmicro final data sheets supply the characterization information on the transconductance of the oscillator (measurement of gain). This information can be used to check part to part variations of the PICmicro.

When selecting the crystal, the designer must ensure that it is a parallel cut type. Failure to use a parallel cut crystal may cause:

- Frequency operation out of the specified range of the crystal.
- Unreliable oscillator start-up.
- Device or crystal damage.

RAM and Special Function Register Initialization

After a successful Power-up Reset, the device will begin to execute the firmware program. To have expected operation, ALL RAM should be initialized by the program. This includes the Special Function Registers (SFR) and the general purpose data memory. The use (read) of an uninitialized RAM location will cause the program to do exactly what you told it, with the unexpected RAM value. It should not be expected that all devices will power-up with the same uninitialized device values.

There are many factors that contribute to how a RAM cell powers up, but the most common "gotcha" is between the Windowed and OTP device types. Many times a user forgets to cover the window after erasing the Windowed device. When the device is powering up, and the light is able to shine onto the device die, the transistor characteristics will shift. This can cause the device RAM to have a different power-up value than a device where no light can shine onto the die (OTP or covered).

Note:	RAM locations should be initialized before they are used. Use of an uninitialized location will cause proper device operation with the improper values. That is, it will do what you told it to do, not what you wanted it to do.

Valid Operating Voltage Levels

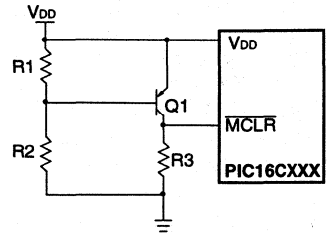
When the device is operating, the device voltage must be within the specified Min/Max limits. Operation of the device outside these limits may cause unexpected device operation.

One of the primary functional failure modes of a device is when the applied voltage is lower than the specified minimum requirement. This functional failure is called Brown-out. Brown-out causes the program memory not to be read correctly. For example, the program counter may be pointing to a `MOVE` instruction, but the device reads it as a `GOTO` instruction (with a random destination). This can have disastrous affects to the operation of the application. If brown-out conditions are possible, the application needs to be protected by using a brown-out circuit. A brown-out circuit works with the $\overline{MCLR}$ pin to put the device in RESET before the device's actual voltage violates the minimum limit.

Figure 13 shows a low cost external brown-out protection circuit. The voltage at which the circuit causes a reset is dependent upon the tolerances of the components. Figure 14 shows the use of a Dallas Semiconductor EconoReset. This device monitors the status of the power supply, and generates a reset when an out-of-tolerance condition is detected. Motorola also makes some 3-terminal devices to monitor the power supply, such as the MC34164, MC34064, MC33064. Their data sheets should be reviewed to ensure that the device is suitable for your application.

© 1997 Microchip Technology Inc.

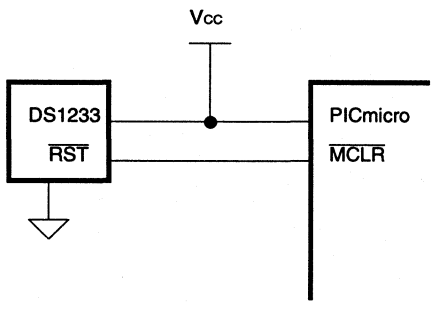
FIGURE 13: LOW COST EXTERNAL BROWN-OUT PROTECTION CIRCUIT



Note: This brown-out circuit is less expensive, albeit less accurate. Transistor Q1 turns off when VDD is below a certain level such that:

$$V_{DD} \bullet \frac{R1}{R1 + R2} = 0.7V$$

FIGURE 14: VOLTAGE SUPERVISORY CHIP



Brown-out and the WDT

The recommended solution for brown-out conditions is the use of a brown-out circuit. The brown-out circuit will keep the device in reset until a valid operating voltage is present. In some applications the additional cost of the external brown-out circuit, can be traded-off with system recovery from brown-out. Use of the Watchdog Timer (WDT) can enhance the probability of system recovery from a brown-out condition.

Note: If I/O drive conflicts can cause critical problems, this technique should not be used. This is due to the indeterminate time before a device reset could occur, which would reset all pins to inputs to eliminate any I/O conflict.

When using the WDT in brown-out conditions, care must be taken. Brown-outs may cause an unrecoverable condition, but with good design practice the probability of this can be significantly reduced.

During a brown-out, improper program execution can occur due to an EPROM read failure. This program execution can also corrupt data memory locations, which include the Special Function Registers (SFRs). Corrupting the control registers may cause hardware conflicts. For example, an input may become an output. Other conflicts are possible, but the situation will be application dependent.

As the device voltage gets lower, internal logic can become corrupted. This can include the Program Counter (PC) value, Stack Pointer and contents, State machines, Data Memory, etc.

When a valid voltage is returned, the device may be at an unexpected program location, possibly using corrupted values. In this situation, the device would not be expected to operate as intended and could get into a state that appears locked-up.

For the PIC17C42 in code protected microcontroller mode, once the Program Counter (PC) exceeds the 32K-word boundary, the device will become locked-up. The PC can exceed the 32K-word boundary from the execution of incorrect instructions (due to failure reading the EPROM) or by the PC becoming corrupted.

If the WDT is to be used to reset the device, care must be used in structuring the program. Optimally, only one CLRWDT instruction should be used. This minimizes the possibility of program execution returning to a loop which clears the WDT. This loop could then lock-up the device, since other control registers are corrupted and the device is not configured as expected. An example is; if the loop was waiting for an interrupt, but the bit that enables global interrupts was disabled, the device would no longer respond to the interrupts and would appear locked-up.

Example 1 shows a simple implementation of using the WDT reset for system recovery. The program loops, waiting for a WDT time-out (which clears the $\overline{TO}$ bit). After the WDT reset, the $\overline{TO}$ bit needs to be set (by executing a CLRWDT instruction). The program should then initialize the device. Then application code can start executing. There is a possibility of the $\overline{TO}$ bit being corrupted by low voltage, and the device not being in a reset state when the software initializes the device.

The WDT example in Appendix B: uses a different method, independent of the $\overline{TO}$ bit. This uses RAM locations which get loaded with a value. A WDT time-out (or other reset) needs to occur. The RAM locations are verified to contain the same values. Once the RAM is verified, it is cleared, and the device should be initialized. These RAM locations can be used by the application program.

© 1997 Microchip Technology Inc.

EXAMPLE 1: USING WDT RESET

```
org     Reset_Address
        GOTO    TO_TEST     ;At any reset,
                            ;test the TO bit
org     TO_TEST
        BTFSC   STATUS, TO  ;WDT Time-Out?
HERE    GOTO    HERE        ;NO, Wait for TO
Time_Out                    ;YES, Good Reset
        CLRWDT              ;Start here
        :                   ;Initialize Device
        :                   ;Application Code
```

False Power-down

In applications where power is removed from the device's supply lines, but voltage is still applied to an I/O pin, unexpected operation may occur. Power is able to be supplied to the device through this I/O pin. Since the device is still partially powered, the internal logic is never completely powered down. Figure 15 shows the general structure of an I/O pin. Figure 16 depicts the internal voltage level that is actually applied to some device logic, versus what is seen at the pin.

To guarantee a Power-on Reset (POR) rising edge, the device voltage (V_{DD}) must start from Vss. When the device is inadvertently powered from an I/O pin, the voltage at the V_{DD} pin may appear to be near ground but may actually be higher in the device. With some of the internal logic powered, the characteristics of the device can be similar to a brown-out situation. Similar design practices to brown-out should be implemented.

A method for protecting the device from being powered from an I/O pin is shown in Figure 17.

FIGURE 15: TYPICAL ELECTRICAL STRUCTURE OF I/O PIN

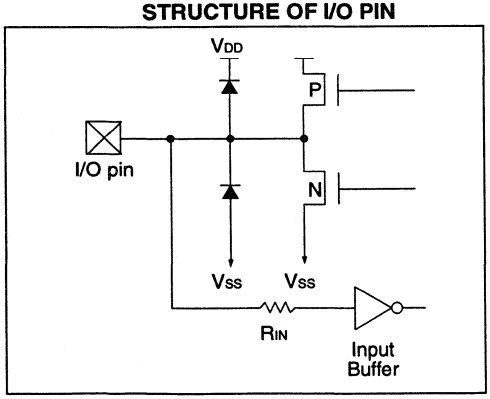


FIGURE 16: FALSE POWER-DOWN

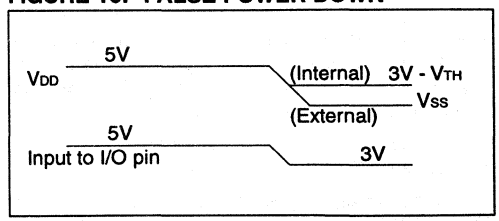
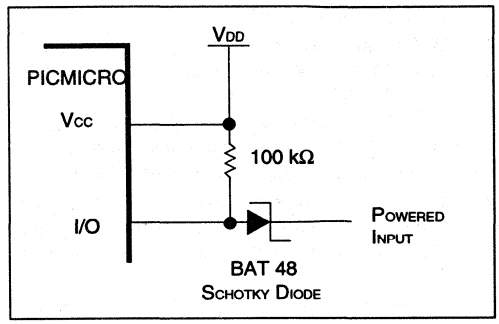


FIGURE 17: POWERED INPUT PROTECTION



In general, a brown-out detect circuit should cause the PICmicro to RESET ($\overline{MCLR}$ forced low). This ensures that the internal logic is in a known state until a valid device voltage level is reached. The actual brown-out circuit depends on the voltage range of the device and the application requirements. A comprehensive brown-out circuit would use a dedicated device to monitor the voltage and force the $\overline{MCLR}$ pin low when the voltage becomes lower than specified.

Another case of false power-down situations is when the power is removed from the system, but the capacitor loading keeps a non-zero voltage on the V_{DD} pin. When power is reapplied, the device never powered down so no Power-on-Reset will occur. A simple brown-out circuit should fix this.

5

TROUBLESHOOTING

There are several techniques that can be used to troubleshoot problems related to powering up. First it is important to try to locate the source of the problem. These sources could be:

- No oscillation on OSC1/OSC2 pins
- Improper/no Program Execution

In cases where there is no oscillation on the OSC1/OSC2 pins, some of the following should be tried:

a) Verify that there are good connections/the components are good.
b) Verify that the crystal/resonator manufacturer is one that has been tested, if not try other capacitor values.
c) See if an external clock (from a function generator) causes device operation to begin.
d) Verify that all components are well grounded.
e) If a scope probe is connected to the oscillator output, it must be a low capacitance/high impedance probe. If it is not, the oscillator may stop.

In cases where program execution is not as expected:

a) Use a minimal program with external clock input.
b) Tie $\overline{MCLR}$ to ground until solid power is applied to the device then release $\overline{MCLR}$ (bring high).
c) Measure V_{DD} rise time to determine if an external reset circuit is needed, and, if so, what type of reset circuit should be used.
d) Verify that the device program memory and configuration bits are programmed to their expected states.

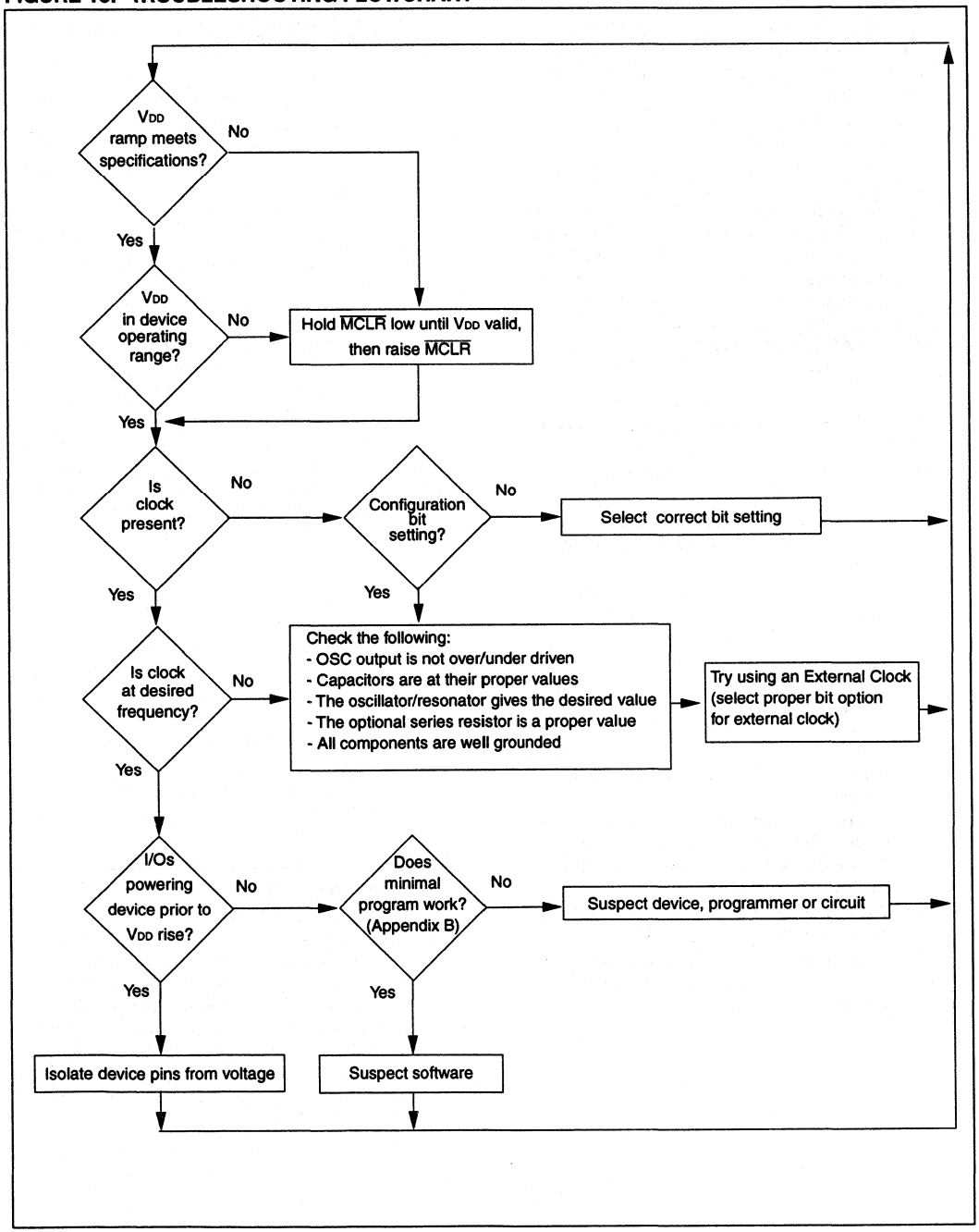
The flowchart shown in Figure 18 can be used to troubleshoot power-up problems. This flowchart is only intended to be the first level diagnostic in trying to solve a power-up problem. Many other flowcharts can be used, depending on the characteristics of the problem and the set-up of the application.

CONCLUSION

Understanding the criteria for the powering up of a device will allow you to make better design choices. If device power-up problems are still encountered, many techniques can be used to solve the problem. Appendix B contains example code which can be used to verify that a device is operating (powered-up correctly). This eliminates the possibility of the program as the cause, and allows debug on the hardware.

© 1997 Microchip Technology Inc.

FIGURE 18: TROUBLESHOOTING FLOWCHART



5

APPENDIX A: Q & As

Q. ***When I use a windowed device (JW), my application works as expected. When I program an OTP device, it no longer works as expected. Why is this?***

A. The silicon is the same between the OTP and windowed devices. If the windowed device's window is not covered (with black tape), light shines onto the silicon. The light causes the potential levels of gates to shift. This in turn can cause RAM to be initialized to an unknown state, which could be different than in the OTP device. If RAM is not initialized by the program before it is used, these different power-up states of the RAM could be the cause of the problem. Ensure that all RAM is initialized in the device. This includes the SFRs.

Q. ***My oscillator is not oscillating, what could be wrong?***

A. There are several possibilities, some which include:

1. The wrong oscillator bit setting is selected. The erased (default) state is RC oscillator mode.
2. The wrong capacitor values are installed. Refer to the most current data sheet for recommended values.
3. The characteristics of your manufacturers crystal are different than those that are characterized by Microchip. Generally our tests have been done with one of the following manufacturers' crystals/ resonators: ECS, FOX, Murata Erie, or Panasonic.
4. The external connections to the device are wrong. Verify that all connections to the device are correct and that good signals / levels are being applied.
5. The cut of the crystal is a series type, as opposed to the specified parallel type.
6. No bypassing capacitors were used on the device. The noise on V_{DD} could be affecting the oscillator circuitry.

Q. ***The device was powered-down and then powered back up, but the device does not operate. What could be wrong.***

A. Possibilities include:

1. If power was applied to an I/O pin when the device was "powered-down", the device would be powered through the I/O pin. The internal logic is not actually powered-down, and Power-on Reset (POR) will not occur.
2. When V_{DD} was powered-down, V_{DD} was not given enough time to settle to 0V.
3. The V_{DD} ramp rate is too slow.

Q. ***My oscillator is oscillating, but the device is not working. What could be wrong?***

A. There are several possibilities, some which include:

1. Slow V_{DD} rise time, which was too slow to cause a Power-on Reset (POR). The rise time should not exceed the minimum device specification. For most devices this is 0.05 V/ms. Also the device must be at the minimum operating V_{DD} of the processor when reset is exited.
2. Ensure that the $\overline{MCLR}$ pin is not low. This holds the device in RESET.
3. A brown-out has occurred, and has corrupted the internal state machines (including the WDT). An external brown-out circuit is recommended to hold the device in RESET during the brown-out condition.
4. The `CLRWDT` instruction is not being used (often enough) when the WDT is enabled.

Q. ***When I power-up the device, it does not operate and it gets hot.***

A. Your design is probably permitting fast high voltage signals (spike) onto one of the device pins. This sudden high voltage (and associated current) is in excess of the protection diode limit. The device must be powered-down (to V_{SS}) to release this condition. This condition may cause a functional failure or affect device reliability. All Microchip devices meet or exceed the Human Body Model (HBM) and Machine Model (MM) for ESD and latch-up.

© 1997 Microchip Technology Inc.

Q. ***My oscillator is oscillating, but not at the expected frequency. What could be wrong.***

A. For many designers, working with oscillators and their related issues are a "black magic", since the characteristics can vary widely between manufacturers. I suggest that you read all the application notes that we have available on oscillators. Some quick possibilities are:

1. The cut of the crystal is a series type, as opposed to the specified parallel type.
2. No bypassing capacitors were used on the device. The noise on V_{DD} could affect the oscillator circuitry.
3. The capacitor values used are causing the oscillator to operate in one of the harmonic frequencies.

Note: This is not an all inclusive list. You may need to investigate other design aspects.

Q. ***The device seems to never exit reset, or is continually resetting.***

A. The CLRWDT instruction is not being used (often enough) when the WDT is enabled.

Q. ***The device was powered-down and back up again, but it does not reset. It just starts operating immediately.***

A. Possibilities include:

1. If power was applied to an I/O pin when the device was "powered-down", the device would be powered through the I/O pin. The internal logic is not actually powered-down, and a Power-on Reset (POR) will not occur.
2. When V_{DD} was powered down, V_{DD} was not given enough time to settle to 0V.

Q. ***The oscillator is operating (I check it with a scope), yet when I look at other pins the program is not executing. Why?***

A. One possible reason is that when the oscilloscope probe is placed on the OSC2 pin, the additional capacitance is enough to cause oscillation to start. Removing the capacitive load of the probe causes the oscillation to stop.

5

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: TEST PROGRAMS

PIC16C5X BIT TOGGLE

```
MPASM 01.02.04 Intermediate   C5X_B0T.ASM   12-20-1994  9:25:7                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     0001          LIST    P = 16C54,  F = INHX8M, n = 66
                     0002 ;
                     0003 ;*****************************************************************************
                     0004 ;
                     0005 ; This program is a minimum program to toggle a single I/O port pin for the
                     0006 ; 16C5x family of devices. The only initialization is that of the data
                     0007 ; direction register (TRIS) of the I/O pin and the Toggling of the pin.
                     0008 ; The waveform will be 1 unit high and 3 units low.
                     0009 ;
                     0010 ;        Program:          C5X_B0T.ASM
                     0011 ;        Revision Date:    12-20-94
                     0012 ;
                     0013 ;*****************************************************************************
                     0014 ;
                     0015 ;
                     0016 ; HARDWARE SETUP
                     0017 ;        None
                     0018 ;
                     0019 ;
                     0020              INCLUDE <p16C5x.inc>
                     0002 ; P16C5X.INC  Standard Header File, Version 0.1     Microchip Technology, Inc.
                     0020
                     0021 ;
  0FF9               0022          __FUSES ( _CP_OFF & _WDT_OFF & _XT_OSC )
                     0023 ;
                     0024 ;*****************************************************************************
                     0025 ;*****       Start program here.
                     0026 ;*****************************************************************************
                     0027 ;
0000                 0028 START                                  ; POWER_ON Reset (Beginning of program)
0000 0063            0029              CLRF    STATUS            ; Do initialization (Bank 0)
0001 0C00            0030              MOVLW   0x00              ; Specify value for PortB output latch
0002 0026            0031              MOVWF   PORTB             ;
0003 0C00            0032              MOVLW   0x00              ; Specify which PortB pins are inputs / outputs
0004 0006            0033              TRIS    PORTB             ;
                     0034 ;
0005 0506            0035 lzz          BSF     PORTB, 0          ; B0 is High
0006 0406            0036              BCF     PORTB, 0          ; B0 is Low
0007 0A05            0037              GOTO    lzz               ; Loop
                     0038 ;
                     0039 ;
                     0040
                     0041 ;
                     0042 ; Reset address. Determine type of RESET
                     0043 ;
                     0044      IFDEF   __16C54
  01FF               0045 RESET_V     EQU     0x1FF
                     0046      ENDIF
                     0047 ;
                     0048      IFDEF   __16C54A
                     0049 RESET_V     EQU     0x1FF
                     0050      ENDIF
                     0051 ;
                     0052      IFDEF   __16C55
                     0053 RESET_V     EQU     0x1FF
                     0054      ENDIF
                     0055 ;
                     0056      IFDEF   __16C56
                     0057 RESET_V     EQU     0x3FF
                     0058      ENDIF
                     0059 ;
                     0060      IFDEF       __16C57
                     0061 RESET_V         EQU     0x7FF
                     0062      ENDIF
```


© 1997 Microchip Technology Inc.

```
                    0063 ;
                    0064     IFDEF        __16C58A
                    0065 RESET_V          EQU     0x7FF
                    0066     ENDIF
                    0067 ;
  01FE              0068 PROG_MEM_END     EQU     RESET_V - 1
                    0069 ;
                    0070 ;
                    0071         org      PROG_MEM_END        ; End of Program Memory
01FE 0BFE           0072 ERR_LP_1     GOTO     ERR_LP_1        ; If you get here your program was lost
                    0073 ;
                    0074         org      RESET_V             ; RESET vector location
01FF 0A00           0075 R_VECTOR     GOTO     START           ;
                    0076 ;
                    0077 ;
                    0078     end
                    0079
                    0080
                    0081


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXX-------- ---------------- ---------------- ----------------
0040 : ---------------- ---------------- ---------------- ----------------

0180 : ---------------- ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- --------------XX

All other memory blocks unused.


Errors   :     0
Warnings :     0
Messages :     0
```

5

PIC16CXXX BIT TOGGLE

```
MPASM 01.02.04 Intermediate   CXX_B0T.ASM   12-20-1994  10:18:22                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     0001         LIST    P = 16C74,  F = INHX8M, n = 66
                     0002 ;
                     0003 ;*****************************************************************************
                     0004 ;
                     0005 ; This program is a minimum program to toggle a single I/O port pin for the
                     0006 ; 16Cxx family of devices. The only initialization is that of the data
                     0007 ; direction register (TRIS) of the I/O pin and the Toggling of the pin.
                     0008 ; The waveform will be 1 unit high and 3 units low.
                     0009 ;
                     0010 ;       Program:         CXX_B0T.ASM
                     0011 ;       Revision Date:   12-20-94
                     0012 ;
                     0013 ;*****************************************************************************
                     0014 ;
                     0015 ;
                     0016 ; HARDWARE SETUP
                     0017 ;       None
                     0018 ;
                     0019 ;
                     0020             INCLUDE <p16Cxx.inc>
                     0002 ; P16CXX.INC  Standard Header File, Version 0.2     Microchip Technology, Inc.
                     0020
                     0021 ;
  3FF9               0022         __FUSES ( _CP_OFF & _WDT_OFF & _XT_OSC & _PWRTE_ON )
                     0023 ;
                     0024 ;*****************************************************************************
                     0025 ;*****      Start program here.
                     0026 ;*****************************************************************************
                     0027 ;
0000                 0028 START                                ; POWER_ON Reset (Beginning of program)
0000 0183            0029             CLRF    STATUS           ; Do initialization (Bank 0)
0001 3000            0030             MOVLW   0x00             ; Specify value for PortB output latch
0002 0086            0031             MOVWF   PORTB            ;
0003 1683            0032             BSF     STATUS, RP0      ; Bank 1
0004 3000            0033             MOVLW   0x00             ; Specify which PortB pins are inputs / outputs
0005 0086            0034             MOVWF   TRISB            ;
0006 1283            0035             BCF     STATUS, RP0      ; Bank 0
                     0036 ;
0007 1406            0037 lzz         BSF     PORTB, 0         ; B0 is High
0008 1006            0038             BCF     PORTB, 0         ; B0 is Low
0009 2807            0039             GOTO    lzz              ; Loop
                     0040 ;
                     0041 ;
                     0042
                     0043 ;
                     0044 ; End of Program Memory
                     0045 ;
                     0046     IFDEF   __16C71
                     0047 PROG_MEM_END    EQU     0x3FF
                     0048     ENDIF
                     0049 ;
                     0050     IFDEF   __16C71A
                     0051 PROG_MEM_END    EQU     0x3FF
                     0052     ENDIF
                     0053 ;
                     0054     IFDEF   __16C73
                     0055 PROG_MEM_END    EQU     0xFFF
                     0056     ENDIF
                     0057 ;
                     0058     IFDEF   __16C74
  0FFF               0059 PROG_MEM_END    EQU     0xFFF
                     0060     ENDIF
                     0061 ;
                     0062     IFDEF   __16C61
                     0063 PROG_MEM_END    EQU     0x3FF
                     0064     ENDIF
                     0065 ;
                     0066     IFDEF   __16C63
                     0067 PROG_MEM_END    EQU     0x7FF
                     0068     ENDIF
                     0069 ;
                     0070     IFDEF   __16C64
```


© 1997 Microchip Technology Inc.

```
                    0071 PROG_MEM_END    EQU     0x7FF
                    0072     ENDIF
                    0073 ;
                    0074     IFDEF   __16C65
                    0075 PROG_MEM_END    EQU     0xFFF
                    0076     ENDIF
                    0077 ;
                    0078     IFDEF   __16C84
                    0079 PROG_MEM_END    EQU     0x3FF
                    0080     ENDIF
                    0081 ;
                    0082     IFDEF   __16C84A
                    0083 PROG_MEM_END    EQU     0x3FF
                    0084     ENDIF
                    0085 ;
                    0086 ;
                    0087         org     PROG_MEM_END        ; End of Program Memory
0FFF 2FFF           0088 ERR_LP_1    GOTO    ERR_LP_1        ; If you get here your program was lost
                    0089 ;
                    0090 ;
                    0091     end
                    0092
                    0093
                    0094
                    0095


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXX------ ---------------- ---------------- ----------------
0040 : ---------------- ---------------- ---------------- ----------------

0F80 : ---------------- ---------------- ---------------- ----------------
0FC0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.


Errors   :     0
Warnings :     0
Messages :     0
```

Note: Special Function Register data memory locations, in Bank 1, are specified by their true address in the file PIC16CXXX.INC. The use of the MPASM assembler will generate a warning message, when those labels are used with direct addressing. Warning messages can be turned off with an assembler option.

PIC17CXXX BIT TOGGLE

```
MPASM 01.02.04 Intermediate   P17_B0T.ASM   12-19-1994  17:15:3                PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     0001          LIST    P = 17C42,  F = INHX32, n = 66
                     0002 ;
                     0003 ;*****************************************************************************
                     0004 ;
                     0005 ; This program is a minimum program to toggle a single I/O port pin for the
                     0006 ; 17Cxx family of devices. The only initialization is that of the data
                     0007 ; direction register (DDR) of the I/O pin and the Toggling of the pin.
                     0008 ; The waveform will be 1 unit high and 1 unit low.
                     0009 ;
                     0010 ;       Program:         P17_B0T.ASM
                     0011 ;       Revision Date:   12-20-94
                     0012 ;
                     0013 ;*****************************************************************************
                     0014 ;
                     0015 ;
                     0016 ; HARDWARE SETUP
                     0017 ;       None
                     0018 ;
                     0019 ;
                     0020              INCLUDE <p17Cxx.inc>
                     0002 ; P17CXX.INC  Standard Header File, Version 0.2     Microchip Technology, Inc.
                     0020
                     0021 ;
  FFE2               0022          __FUSES ( _MC_MODE & _WDT_NORM & _XT_OSC )
                     0023 ;
                     0024 ;*****************************************************************************
                     0025 ;*****      Start program here.
                     0026 ;*****************************************************************************
                     0027 ;
0000                 0028 START                                ; POWER_ON Reset (Beginning of program)
0000 2904            0029              CLRF    ALUSTA          ; Do initialization
0001 290F            0030              CLRF    BSR             ; Bank 0
0002 B000            0031              MOVLW   0x00            ; Specify value for PortB output latch
0003 0112            0032              MOVWF   PORTB           ;
0004 B000            0033              MOVLW   0x00            ; Specify which PortB pins are inputs / outputs
0005 0111            0034              MOVWF   DDRB            ;
                     0035 ;
0006 3812            0036 lzz          BTG     PORTB, 0        ; Toggle level on B0
0007 C006            0037              GOTO    lzz             ; Loop
                     0038 ;
                     0039 ;
                     0040
                     0041 ;
                     0042 ; End of Program Memory
                     0043 ;
                     0044      IFDEF   __17C42
  07FF               0045 PROG_MEM_END     EQU     0x7FF
                     0046      ENDIF
                     0047 ;
                     0048 ;
                     0049          org     PROG_MEM_END        ; End of Program Memory
07FF C7FF            0050 ERR_LP_1     GOTO    ERR_LP_1        ; If you get here your program was lost
                     0051 ;
                     0052 ;
                     0053      end
                     0054
                     0055

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXX-------- ---------------- ---------------- ----------------
0040 : ---------------- ---------------- ---------------- ----------------

0780 : ---------------- ---------------- ---------------- ----------------
07C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.


Errors   :     0
Warnings :     0
Messages :     0
```


© 1997 Microchip Technology Inc.

WDT RESET WITH RAM VERIFY

```
MPASM 01.20 Released          BO_RAMT.ASM   6-30-1995  16:04:36         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST    P = 17C44,  F = INHX32, n = 66
                     00002 ;
                     00003 ;*****************************************************************************
                     00004 ;
                     00005 ; This program is a minimum program to recover from a brown-out condition thru
                     00006 ; the use of the WDT. The method is to load RAM locations with a known value
                     00007 ; and compare these locations after any RESET. If the RAM location matches the
                     00008 ; expected value then program flow can continue. The longer this RAM string
                     00009 ; is, the greater the probability that the RAM would NOT power up in that state.
                     00010 ;
                     00011 ;
                     00012 ; NOTE: This does not Guarantee device recovery, due to the random start-up
                     00013 ;       point after brown-out. This point could be a loop with a CLRWDT
                     00014 ;       instruction. The recommended solution is to always use a brown-out
                     00015 ;       circuit.
                     00016 ;
                     00017 ;       Program:         BO_RAMT.ASM
                     00018 ;       Revision Date:   06-29-95
                     00019 ;
                     00020 ;*****************************************************************************
                     00021 ;
                     00022 ;
                     00023 ; HARDWARE SETUP
                     00024 ;       None
                     00025 ;
                     00026 ;
  0001               00027 TRUE        EQU    1
  0000               00028 FALSE       EQU    0
                     00029 ;
  0001               00030 Debug       EQU    TRUE
                     00031 #define     __CONFIG __FUSES
                     00032 ;
                     00033             INCLUDE <DEV_FAM.inc>
                     00102     list
                     00034 ;
                     00035     if ( P16C5X )
                     00036             INCLUDE <p16C5x.inc>
                     00037         __CONFIG ( _CP_OFF & _WDT_ON & _XT_OSC )
                     00038     endif
                     00039 ;
                     00040     if ( P16CXX )
                     00041             INCLUDE <p16Cxx.inc>
                     00042         __CONFIG ( _CP_OFF & _WDT_ON & _XT_OSC & _PWRTE_ON )
                     00043     endif
                     00044 ;
                     00045     if ( P17CXX )
                     00046             INCLUDE <p17Cxx.inc>
                     00001         LIST
                     00002 ; P17CXX.INC  Standard Header File, Version 2.01     Microchip Technology, Inc.
                     00298         LIST
FE00 FFE2            00047         __CONFIG ( _MC_MODE & _WDT_NORM & _XT_OSC )
                     00048     endif
                     00049 ;
                     00050     if ( P16C5X + P16CXX + P17CXX != 1 )
                     00051 MESSG  "WARNING - USER DEFINED: One and only one device family can be selected"
                     00052     endif
                     00053 ;
                     00054             INCLUDE <BO_RAMT.inc>
                     00029     list
                     00055             INCLUDE <PMEM_END.inc>
                     00116     list
                     00056 ;
                     00057 ;
                     00058 ;*****************************************************************************
                     00059 ;*****      Start program here.
                     00060 ;*****************************************************************************
                     00061 ;
0000                 00062     org  Reset_Address
                     00063 ;                               ; in the LIST directive
                     00064     if ( P16C5X )
                     00065     org  0h                     ; Override the start of this code.
                     00066         CLRF   STATUS           ; Force program memory to Page 0
                     00067         CLRF   FSR              ; Force Data Memory to Bank 0
                     00068     endif
```

```
                    00069 ;
                    00070     if ( P16CXX )
                    00071         CLRF    PCLATH          ; Force program memory to Page 0
                    00072         CLRF    STATUS          ; Force Data Memory to Bank 0
                    00073     endif
                    00074 ;
                    00075     if ( P17CXX )
0000 2903           00076         CLRF    PCLATH, F       ; Force program memory to Page 0
0001 290F           00077         CLRF    BSR, F          ; Force Peripheral / GP Data Memory to Bank 0
                    00078     endif
                    00079 ;
0002 C100           00080         GOTO    RAM_TEST        ; At any reset,
                    00081                                 ;  test the RAM
                    00082 ;
                    00083 ; In RAM_TEST, program execution is held-off until a valid "warm" reset
                    00084 ; occurs. That is, the contents of some RAM locations retain the
                    00085 ; values that were written to them. The probability that the RAM would power-up
                    00086 ; in that state is dependent on the number of bytes of RAM used. The
                    00087 ; more RAM, the less the probability (probability = 1 / ( 2 ** 8(N+1) ).
                    00088 ;
                    00089 ;
0100                00090     org  MAIN                   ; In Program Memory Page 0
0100                00091 RAM_TEST
0100 B0A5           00092         MOVLW  BYTE_0
0101 0520           00093         SUBWF  RAM0, F
0102 9204           00094         BTFSS  STATUS, Z        ; Result = 0?
0103 C110           00095         GOTO   LD_RAM           ; NO, Load Ram
                    00096 ;
0104 B00F           00097         MOVLW  BYTE_1           ; YES, Check next
0105 0521           00098         SUBWF  RAM1, F          ;   location
0106 9204           00099         BTFSS  STATUS, Z        ; Result = 0?
0107 C110           00100         GOTO   LD_RAM           ; NO, Load RAM
                    00101 ;
                    00102 ;       :                       ; YES, Do Again
                    00103 ;       :                       ;
                    00104 ;
0108 B05A           00105         MOVLW  BYTE_n           ; YES, Check nth
0109 0522           00106         SUBWF  RAMn, F          ;   location
010A 9204           00107         BTFSS  STATUS, Z        ; Result = 0?
010B C110           00108         GOTO   LD_RAM           ; NO, Load RAM
                    00109 ;
                    00110     if ( P16C5X || P16CXX )
                    00111         CLRF   RAM0             ; YES, Time-out
                    00112         CLRF   RAM1             ;   occured, clear
                    00113 ;       :                       ;   RAM locations
                    00114 ;       :
                    00115         CLRF   RAMn             ;
                    00116     endif
                    00117 ;
                    00118 ;
                    00119     if ( P17CXX )
010C 2920           00120         CLRF   RAM0, F          ; YES, Time-out
010D 2921           00121         CLRF   RAM1, F          ;   occurred, clear
                    00122 ;       :                       ;   RAM locations
                    00123 ;       :
010E 2922           00124         CLRF   RAMn, F          ;
                    00125     endif
                    00126 ;
010F C117           00127         GOTO   Time_Out         ; Initialize Device
                    00128
0110                00129 LD_RAM
0110 B0A5           00130         MOVLW  BYTE_0           ; Load RAM
0111 0120           00131         MOVWF  RAM0             ;   locations to
0112 B00F           00132         MOVLW  BYTE_1           ;   compare against
0113 0121           00133         MOVWF  RAM1             ;
                    00134 ;       :
0114 B05A           00135         MOVLW  BYTE_n           ;
0115 0122           00136         MOVWF  RAMn             ;
                    00137 ;
0116 C116           00138 HERE    GOTO   HERE             ; Wait for WDT TO
0117                00139 Time_Out                        ; YES, Good Reset
0117 0004           00140         CLRWDT                  ;   Start here
                    00141 ;       :                       ; Initialze Device
                    00142 ;       :                       ;  Application Code
                    00143 ;
                    00144     if ( Debug )                ;
                    00145         if ( P16C5X )
                    00146             CLRF    PORTB       ; PORTB output latch is cleared
                    00147             MOVLW   0x00        ;
                    00148             TRIS    PORTB       ; Port B is output
                    00149             BCF     PORTB, 0    ;
                    00150             BSF     PORTB, 0    ; Toggle pin B0
                    00151         endif
```


© 1997 Microchip Technology Inc.

```
                    00152 ;
                    00153         if ( P16CXX )
                    00154             CLRF    PORTB         ; PORTB output latch is cleared
                    00155             BSF     STATUS, RP0   ; Bank 1
                    00156             CLRF    TRISB         ; Port B is output
                    00157             BCF     STATUS, RP0   ; Bank 0
                    00158             BCF     PORTB, 0      ;
                    00159             BSF     PORTB, 0      ; Toggle pin B0
                    00160         endif
                    00161 ;
                    00162         if ( P17CXX )
0118 2912           00163             CLRF    PORTB, F      ; PORTB output latch is cleared
0119 2911           00164             CLRF    DDRB, F       ; Port B is output
011A 8812           00165             BCF     PORTB, 0      ;
011B 8012           00166             BSF     PORTB, 0      ; Toggle pin B0
                    00167         endif
                    00168     endif
                    00169 ;
011C C117           00170         GOTO    Time_Out          ; Return to start of Program
                    00171 ;
1FFF                00172         org     PROG_MEM_END      ; End of Program Memory
1FFF                00173 ERR_LP_1
1FFF DFFF           00174         GOTO    ERR_LP_1          ; If you get here your program was lost
                    00175 ;
                    00176     if ( P16C5X )
                    00177         NOP                       ; This will cause the Program memory rollover
                    00178                                   ;    for PIC16C5x devices
                    00179     endif
                    00180 ;
                    00181 ;
                    00182     end


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXX------------ ---------------- ---------------- ----------------
0040 : ---------------- ---------------- ---------------- ----------------
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXX--- ---------------- ----------------
0140 : ---------------- ---------------- ---------------- ----------------
1F80 : ---------------- ---------------- ---------------- ----------------
1FC0 : ---------------- ---------------- ---------------- ---------------X
FE00 : X--------------- ---------------- ---------------- ----------------
FE40 : ---------------- ---------------- ---------------- ----------------

All other memory blocks unused.


Errors   :     0
Warnings :     0
Messages :     0
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN615

Clock Design Using Low Power/Cost Techniques

Author: *John Day*
Sr., Field Application Engineer, Boston

INTRODUCTION

Typical embedded control applications place demands such as low power consumption, small size, low cost and reduced component count onto the microcontroller. This application note implements a 24-hour digital clock, alarm and 99 minute 59 second count down timer, yet operates on two "AA" batteries. The PIC16C54A is perfect for this application, due to it's small size, high current I/Os with direct LED drive, low cost, fast instruction throughput and low frequency/current operation.

System cost

The objective of this design was to implement the maximum number of features with the least expensive and smallest device. The PIC16C54A is Microchip's lowest cost microcontroller and it has 12 I/O lines, each capable of sinking 25 mA and sourcing 20 mA. High efficiency common cathode LED displays were chosen for their 3.5 mA current requirement, eliminating the need for any external transistors for display drive. A low impedance direct drive piezo buzzer was chosen and it's tone is generated by the software of the PIC16C54A to further reduce system cost.

Operating power

In battery powered applications, the operating current determines the lifetime of the batteries. There are many ways to reduce the operating current of any application, including low frequency operation and the use of sleep mode. Since the clock has to keep track of time, SLEEP mode could not be used and the processor must be kept running all of the time. The PIC16C54A supports the 32.768 kHz "watch" crystal and typically consumes less than 15 µA of current in this configuration. Since the PIC16C54A executes instructions in one cycle and it's instruction set is very efficient, this application was able to be implemented using a low frequency crystal. Another solution to this problem comes with the PIC16C74/73/65/63 in it's Timer1 module. Timer1 will run when the device is asleep, so it could have been used to keep track of time, simplifying the software.

Clock system

A 32.768 kHz crystal was chosen for the clock due to the low power and cost requirements of this design. The four internal phases of this input clock create an internal instruction cycle. Therefore, the instruction time is calculated as follows:

$$\text{Instruction rate} = \frac{1}{(\text{CLKIN}/4)} = \frac{1}{32{,}768/4} = \frac{1}{8{,}192}$$

Instruction cycle = 122.07 µs

This means that every instruction executes in 122.07 µs or we execute exactly 8,192 instructions per second.

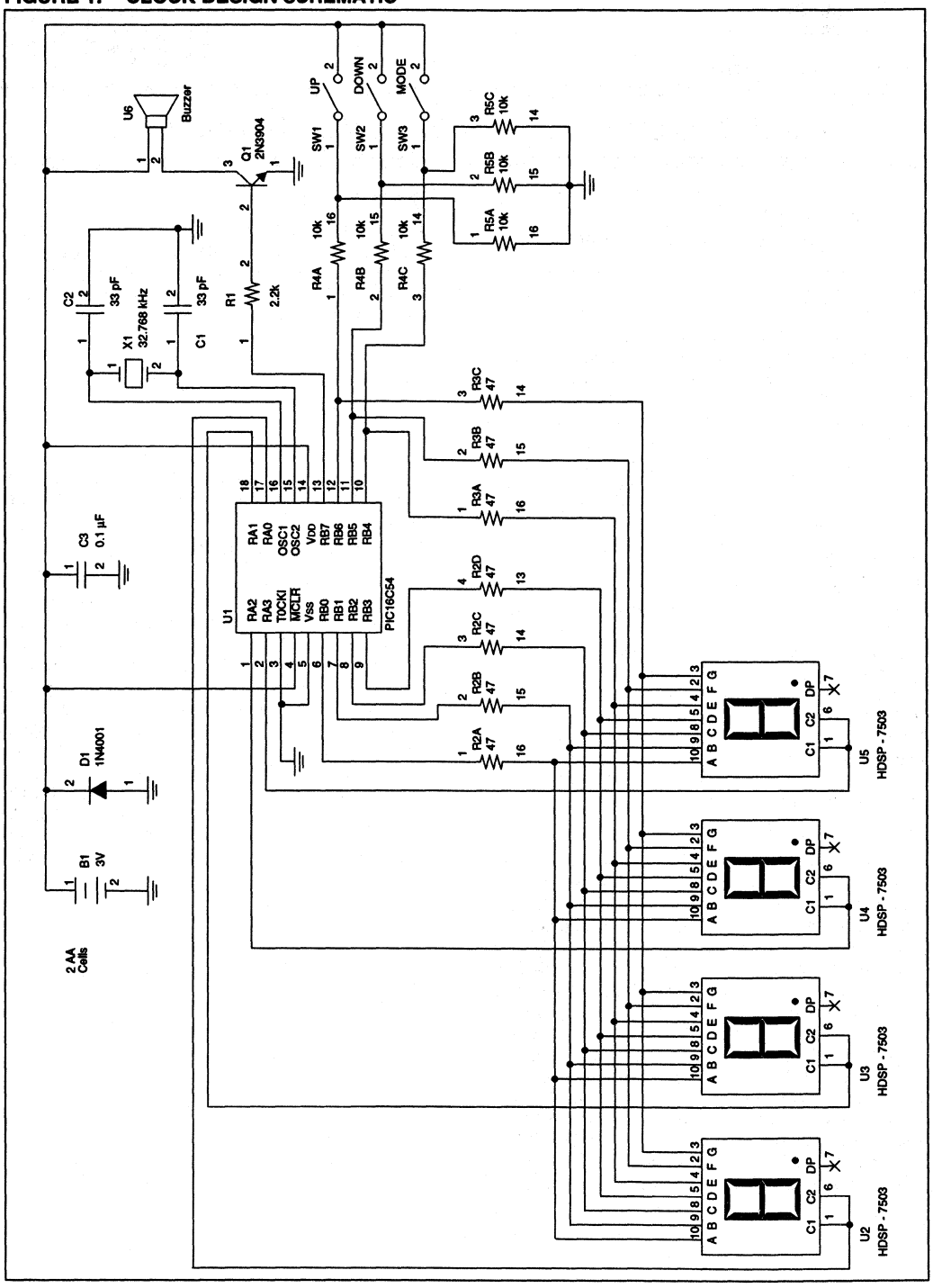
Display and keypad multiplexing

The display contains four digits with seven segments each; therefore a multiplexing scheme was used to reduce the number of I/O lines needed to drive the displays. There are 4 common cathode display connections (one for each display digit and connected to PORTA for convenience so that rotates and moves can be used) and 7 segments (connected to PORTB for convenience so that moves can be used) for a total of 11 I/O lines needed for the display. Common Cathode displays were chosen, since the PIC16C54A can sink 5 mA more current than it can source. The last I/O line (RB7) was used to drive the buzzer. The three keys for setting the time are multiplexed onto the LED display segments to eliminate the need for additional I/O lines.

5

© 1997 Microchip Technology Inc.

FIGURE 1: CLOCK DESIGN SCHEMATIC



© 1997 Microchip Technology Inc.

SOFTWARE IMPLEMENTATION

The main loop of the software must perform the following tasks to implement the clock's functionality:

1. Determine when one second has passed (when bit7 on the TMR0 register changes state 4 times), increment the current time and (if enabled) decrement the countdown timer.
2. Determine if any of the alarms (countdown timer or the alarm time) are currently alarming or should be alarming. If so, the buzzer is buzzed.
3. Check for any keys that are pressed. If MODE is pressed, the current mode is incremented and if UP or DOWN is pressed, the time displayed is modified.
4. Automatically turn the displays on/off for power management.
5. Multiplex the displays every 3.9 ms (32 instruction cycles).

General purpose registers are defined and used for the following purposes:

- **DISPSEGS_A** through **DISPSEGS_D** store the bit pattern that is to be displayed on each of the four 7-segment displays.
- **CLK_SEC** stores the second counter for the current clock time (values from 0-59 decimal are stored).
- **CLK_MIN_LD, CLK_MIN_HD** store the upper and lower minute digit of the current time.
- **CLK_HOUR_LD, CLK_HOUR_HD** store the upper and lower hour digit of the current time.
- **ALM_MIN_LD, ALM_MIN_HD** store the upper and lower minute digit of the alarm time.
- **ALM_HOUR_LD, ALM_HOUR_HD** store the upper and lower hour digit of the alarm time.
- **TMR_SEC_LD, TMR_SEC_HD** store the upper and lower second digit of the countdown timer.
- **TMR_MIN_LD, TMR_MIN_HD** store the upper and lower minute digit of the countdown timer.
- **KEYPAT** stores a pattern showing the currently pressed keys:
  - UP = bit6
  - DOWN = bit5
  - MODE = bit4
- **FLAGS** stores key flag bits such as the current mode, display on, alarm on, etc.
- **PREVTMR0** stores previous TMR0 values so that the differences can be detected the next time the TMR0 is polled.
- **TEMP** is a temporary register used for various routines.
- **DISPONCNT** stores the remaining number of seconds the displays should be on.
- **MODE_COUNT** stores the number of 1/2 seconds the MODE and UP or DOWN buttons are pressed. Used to switch from setting minutes to hours.
- **ALARMCNT** stores the number of beeps remaining to be driven into the buzzer.

FLAGS Register

Most designs require flag or state bits to indicate current modes or the state of a software routines. In this design, the FLAGS register is defined as follows:

Bits 0,1 -indicates the current operating mode (changed by pressing and releasing the MODE button):

00 - Display OFF

01 - Display/Set countdown timer

10 - Display/Set alarm time

11 - Display/Set clock (current time)

Bit 3 - indicates if (alarm time) = (current clock time)

Bit 4 - indicates if (count down timer) = 0

Bits 5,6,7 -Used as a divide by four counter to keep track of seconds

The software is broken up into the following routines for modularity:

buzz_now routine - *Output buzzing tone during alarm for 156 ms.*

Buzzers are available in self-oscillating and direct drive models. To save cost, a low impedance, direct drive model was selected. The buzz_now routine is called by the main_loop and it chirps the buzzer for 156 ms at a 1638 Hz frequency. This routine first turns off the LEDs (by clearing PORTB) and then uses TEMP to count for 256 pulses. The pulse is sent to the buzzer by the `BSF BUZZEROUT` and `BCF BUZZEROUT` instructions. This routine returns once 256 pulses are sent to the buzzer. This is necessary, since the controller cannot buzz the buzzer and keep track of time at the same time (running at such a low frequency), so these two functions are multiplexed.

5

© 1997 Microchip Technology Inc.

task_scan routine - *multiplex LEDs to display the next digit, only one digit is lit at a time).*

The PIC16C5X family is designed for polled I/O applications and does not contain a hardware interrupt structure. To achieve the lowest cost design, the PIC16C54A was selected and all modules are written to CALL this task_scan routine within the multiplexing time frame of 3.9 ms or 32 instruction cycles. This routine first synchronizes itself with the TMR0 register, bit0 to ensure that the scanning occurs at the same point in time, regardless of when the routine is called. Next, PREVSCAN is rotated, setting up the CARRY bit correctly. The bit pattern for the next digit to be displayed is then moved into the W. register. The display is blanked, PORTA is rotated (to select the next digit) and the next display bit pattern is moved to PORTB to display it. For ESD integrity, PORTA is later restored from the PREVSCAN register. This routine takes a total of 21 cycles (including the CALL and RETLW instructions) to execute and the displays are scanned every 3.9 ms (32 instruction cycles); therefore, this routine needs to be called after every 11 instruction cycles from every routine to maintain proper display multiplexing.

disp_value routine - *Update the display registers with the bitmap of what digits are to be displayed next.*

Indirect addressing is used here to reduce the amount of code needed and to simplify the routine. Since the clock, alarm or countdown time could each be displayed, the W register contains the base address (in the register file) of the four digits that are to currently be displayed. The W register is first moved to the FSR register so that the indirect address register contains the first digit to be displayed. The first digit is first converted into the segment bit pattern by calling the led_lookup table and then the bit pattern is moved to DISPSEGS_A. The FSR register is incremented (moving to the next digit) and the process is repeated for the remaining 3 digits. To maintain proper multiplexing, task_scan is called throughout this routine.

turnon_scan routine - *Turns on the LEDs and restores a legal scan position.*

To save battery power, the displays are automatically shut off after 8 seconds when no buttons are pressed. The DISPON bit is used to preset the remaining display on time to 8 seconds. This routine sets this flag (to later turn on the displays) and then checks to see if the PREVSCAN register contains a legal value (an illegal value of "FFh" is used to turn off all of the displays) and it restores a legal value if the displays were off.

scan_keys routine - *Turns off LEDs for a moment and scans the push-button inputs.*

To reduce the number of I/Os needed by this application, the three user input keys are multiplexed onto the LED display segments through PORTB. First, the PORTB is cleared and PORTA is set to '0Fh', turning off the LED displays. Next, PORTB is set up with bits 4,5 and 6 as inputs to read the keys. TEMP is then loaded with the keys that have changed state (to detect the falling edge of a key press) and KEYPAT is loaded with a pattern ('0' = not pressed, '1' = pressed) for the keys that are pressed. Lastly, PORTB is restored to all outputs and the current multiplex scan is restored to PORTA.

check_time routine - *Checks for alarm or countdown timer expiration.*

Each second, alarm conditions must be detected and the buzzer sounded if an alarm condition is true. ALARMNOW and EGGNOW are flag bits that are used by the main program to sound the buzzer if they are set. This routine starts by setting both ALARMNOW and EGGNOW. Next, the current time hours and minutes are compared (through a subtraction and a test of the STATUS register Z bit) with the alarm time. If there is any miscompare, the ALARMNOW bit is cleared. To finish, the countdown timer time minutes and seconds digits are each compared with zero. If there is any miscompare, the EGGNOW bit is cleared. To maintain proper multiplexing, the task_scan routine is regularly called throughout this routine.

inc_time routine - *Adds one second, minute or hour to the clock, alarm or timer.*

Every second, inc_time is called by main_loop to increment the seconds count for the clock. This routine is also called when the "UP" key is pressed and "MODE" key is held down to adjust the current time, alarm time or set the countdown timer. This routine uses indirect addressing to reduce the amount of code and simplify it's operation. Before this routine is called, the W register is loaded with the address of the clock second register and the routine is called. The FSR register is loaded with this value and the indirect address register is incremented (effectively incrementing the seconds counter).

Once the second counter is incremented, this register is checked for overflow (greater than 59 seconds) and if no overflow occurred, the routine returns. If an overflow happened, the second counter is cleared and the minute low digit is incremented. This register is then checked for an overflow (greater than 9 minutes) and so on until the all digits are updated.

This routine can also be called from multiple points. If called with the label inc_min_ld, only the minutes (and hours if an overflow occurs) will be incremented. Additionally, calling inc_hour_ld will increment only the hour digits. These features are used when setting the clock or alarm function. The FLAGS register (bits 0 and 1) is used to determine the current mode (clock, alarm or countdown timer) and ensure proper overflow calculations. To maintain proper multiplexing, the task_scan routine is regularly called throughout this routine.

© 1997 Microchip Technology Inc.

dec_time routine - *Subtracts one second, minute or hour from the clock, alarm or timer.*

If the countdown timer is enabled, dec_time is called by the main loop to decrement the seconds count for the countdown timer. This routine is also called by the main loop when the "DOWN" key is pressed and "MODE" key is held down to adjust the current time, alarm time or set the countdown timer. This routine uses indirect addressing to reduce the amount of code and simplify it's operation. Before this routine is called, the W register is loaded with the address of the countdown timer's second register and the routine is called. The FSR register is loaded with this value and the indirect address register is incremented (effectively incrementing the seconds counter).

Once the second counter LSD is decremented, this register is checked for underflow (less than 0 seconds) and if no underflow occurred, the routine returns. If an underflow happened, the second counter LSD is set to 9 and the second MSD is decremented. This register is then checked for an underflow (less than 0 seconds) and so on until all digits are updated.

This routine also can be called from multiple points. If called with the label dec_hour_ld_vec, only the hour digits (or minutes if it is the countdown timer) will be decremented. This feature is used when setting the clock or alarm function. The FLAGS register (bits 0 and 1) is used to determine the current mode (clock, alarm or countdown timer) and ensure proper underflow calculations. To maintain proper multiplexing, the task_scan routine is regularly called throughout this routine.

main_loop routine - *Calls the above routines as needed and keeps track of when to increment the clock or decrement the countdown timer.*

The main_loop calls all of the previous routines as necessary to maintain time, LED multiplexing, alarming and setting each function. The OPTION register is loaded with a 03h value to set up a Divide by 16 prescaler for the TMR0 register and internal instruction cycle increment. The instruction cycle is 122.07 µs; therefore, bit0 changes every (122 µs • 16) = 1.953 ms and bit7 changes every (122.07 µs • 16 • 128) = 250 ms. Bits 5 and 6 of the FLAGS register are used to divide this 250 ms event by 4 to call inc_time every second.

The check_time routine is called after calling inc_time (every second), setting the EGGNOW or ALARMNOW flag bits. If the alarm is enabled, the buzzer is buzzed by calling buzz_now; however, the main timer updates need to occur in between buzzer beeps to keep track of time.

Every 500 ms, the keys are scanned and the edges on the MODE key are detected. Pressing the UP or DOWN key will shut off the buzzer (clearing the enable bits) and pressing the MODE key will advance the current mode. The mode is a 4-state state machine, revolving between the following states:

1. Display OFF - saves battery power - defaults to this mode if no keys are pressed for 8 seconds.
2. Display or Set countdown timer (holding MODE key allows setting).
3. Display or Set Alarm time (holding MODE key allows setting).
4. Display or Set Clock time (holding MODE key allows setting).

Next, the UP and DOWN keyscan values are tested and if the MODE and UP are both pressed, the currently displayed mode time is incremented or decremented. If MODE is not pressed and UP or DOWN is pressed, the displays are turned on, but the displayed time is not altered.

DISPONCNT is used to keep track of how long the displays have been on once all buttons are released. After 8 seconds, the displays are automatically turned off to save power. MODE_COUNT is used to switch from setting the right hand displays (minutes or seconds) to the left hand displays (hours or minutes). When the UP or DOWN button is held with mode for more than 4 seconds consecutively, MODE_COUNT reaches zero, switching from the right to left hand displays.

Finally, the main_loop finishes by updating the display registers by calling disp_value and if DISPONCNT has decreased to zero, the displays are turned off.

Lookup Tables - *Convert a number into a bit pattern or RAM address.*

There are three lookup tables used in this design for BCD to 7-Segment decoding, manufacturing diagnostics and looking up the address of the currently displayed mode.

- **mode_timer** - look-up the address of the clock, Alarm or Timer data storage RAM.
- **led_lookup** - look-up table contains the bitmap display pattern for displaying digits 0-9.
- **mfg_led_lookup** - look-up table contains the bitmap display pattern used for manufacturing mode. Only one segment is lit at a time.

Miscellaneous routines used for initialization and manufacturing test:

- **init** - Initializes all of RAM to zero, sets up the I/O ports and sets default time values.
- **mfg_selftest** - Used in manufacturing mode only - tests each LED segment, push-button, buzzer and display separately to expose bad keys, connections, buzzer or displays.

© 1997 Microchip Technology Inc.

CONCLUSION

The implementation of this application highlights the PIC16C54's highly efficient instruction set, low power and frequency operation, high current direct LED drive capability and high performance instruction execution. Many of the routines used in this application note apply to a variety embedded control applications.

Ram Used: 25 Bytes

Code Space Used: 444 Words (without manufacturing diagnostics)

510 Words (including manufacturing diagnostics)

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: CODE

```
MPASM 01.21.03 Intermediate      CLK8.ASM   8-21-1995  9:17:56         PAGE  1

LOC     OBJECT  LINE SOURCE TEXT
VALUE   CODE

                00001 ; *******************************************
                00002 ; *     PIC Egg Timer Give-Away             *
                00003 ; *                                         *
                00004 ; * Author:   John Day                      *
                00005 ; *           Sr. Field Applications Engineer *
                00006 ; *           Northeast Region              *
                00007 ; *                                         *
                00008 ; * Revision: 1.2                           *
                00009 ; * Date      September 22, 1994            *
                00010 ; * Part:     PIC16C54-LP/P or PIC16LC54A/P *
                00011 ; * Fuses:    OSC:  LP                      *
                00012 ; *           WDT:  OFF                     *
                00013 ; *           Port: OFF                     *
                00014 ; *           CP:   OFF                     *
                00015 ; *******************************************
                00016 ;
                00017 ; This program is intended to run on a 32 Khz watch crystal and
                00018 ; connects to four multiplexed seven segment displays. It displays the
                00019 ; current time, alarm time and egg count down timers. There are
                00020 ; switches that allow the user to set the alarm, timer and clock functions.
                00021
                00022 LIST F=INHX8M,P=16C54
                00023 INCLUDE "p16C5X.inc"
                00001         LIST
                00002 ; P16C5X.INC  Standard Header File, Version 2.02  Microchip Technology, Inc.
                00143         LIST
0FFF    0FF8    00024         __FUSES _CP_OFF&_WDT_OFF&_LP_OSC
                00025
0007            00026         ORG 07h
                00027 ; ***************************************
                00028 ; * Static RAM Register File Definitions *
                00029 ; ***************************************
00000000        00030 INDADDR          EQU     0       ; Indirect address register
00000007        00031 DISPSEGS_A       EQU     07h     ; Current Display A segment bit pattern
00000008        00032 DISPSEGS_B       EQU     08h     ; Current Display B segment bit pattern
00000009        00033 DISPSEGS_C       EQU     09h     ; Current Display C segment bit pattern
0000000A        00034 DISPSEGS_D       EQU     0Ah     ; Current Display D segment bit pattern
0000000B        00035 CLK_SEC          EQU     0Bh     ; Clock second counter (0-59)
0000000C        00036 CLK_MIN_LD       EQU     0Ch     ; Clock minute low digit counter (0-9)
0000000D        00037 CLK_MIN_HD       EQU     0Dh     ; Clock minute high digit counter (0-5)
0000000E        00038 CLK_HOUR_LD      EQU     0Eh     ; Clock hour low digit counter (0-9)
0000000F        00039 CLK_HOUR_HD      EQU     0Fh     ; Clock hour high digit counter (0-2)
00000010        00040 ALM_MIN_LD       EQU     10h     ; Alarm minute low digit counter (0-9)
00000011        00041 ALM_MIN_HD       EQU     11h     ; Alarm minute high digit counter (0-5)
00000012        00042 ALM_HOUR_LD      EQU     12h     ; Alarm hour lor digit counter (0-9)
00000013        00043 ALM_HOUR_HD      EQU     13h     ; Alarm hour high digit counter (0-2)
00000014        00044 TMR_SEC_LD       EQU     14h     ; Timer second low digit counter (0-9)
00000015        00045 TMR_SEC_HD       EQU     15h     ; Timer second high digit counter (0-5)
00000016        00046 TMR_MIN_LD       EQU     16h     ; Timer hour low digit counter (0-9)
00000017        00047 TMR_MIN_HD       EQU     17h     ; Timer hour high digit counter (0-2)
00000018        00048 KEYPAT           EQU     18h     ; Currently pressed key bits
00000019        00049 FLAGS            EQU     19h     ; Status of alarms, display on, etc.
0000001A        00050 PREVTMR0         EQU     1Ah     ; Used to determine which TMR0 bits changed
0000001B        00051 PREVSCAN         EQU     1Bh     ; Store Common Cathode display scan state
0000001C        00052 TEMP             EQU     1Ch     ; Temporary storage
```


© 1997 Microchip Technology Inc.

```
0000001D        00053 DISPONCNT       EQU     1Dh     ; Time the displays have been on
0000001E        00054 MODE_COUNT      EQU     1Eh             ; Current mode state
0000001F        00055 ALARMCNT        EQU     1Fh             ; Time the alarm has been sounding
                00056 ; ************************************
                00057 ; * Flag and state bit definitions    *
                00058 ; ************************************
                00059 #define         SECBIT      TEMP,7      ; Bit to spawn 1/4 second count
                00060 #define         SCANBIT     TMR0,0      ; Bit to spawn display MUX
                00061 #define         MODEKEY     KEYPAT,4    ; Bit for MODEKEY pressed
                00062 #define         UPKEY       KEYPAT,6    ; Bit for UPKEY pressed
                00063 #define         DOWNKEY     KEYPAT,5    ; Bit for DOWNKEY pressed
                00064 #define         MODEKEYCHG  TEMP,4      ; Bit for delta MODEKEY
                00065 #define         TIMENOW     FLAGS,7     ; Flag to indicate 1 second passed
                00066 #define         ALARMNOW    FLAGS,3     ; Flag to indicate wakeup alarm
                00067 #define         EGGNOW      FLAGS,4     ; Flag to indicate egg timer alarm
                00068 #define         ALARMOK     STATUS,PA0  ; Flag to enable wakeup alarm
                00069 #define         EGGOK       STATUS,PA1  ; Flag to enable timer alarm
                00070 #define         BUZZEROUT   PORTB,7     ; Pin for pulsing the buzzer
                00071 #define         DISPON      DISPONCNT,4 ; Bit to turn on LED displays
                00072
                00073 ; ************************************************
                00074 ; * Various Constants used throughout the program *
                00075 ; ************************************************
0000003C        00076 SEC_MAX         EQU     .60             ; Maximum value for second counter
0000000A        00077 MIN_LD_MAX      EQU     .10             ; Maximum value for low digit of minute
00000006        00078 MIN_HD_MAX      EQU     .6              ; Maximum value for high digit of minute
00000004        00079 HOUR_LD_MAX     EQU     .4              ; Maximum value for low digit of hour
00000002        00080 HOUR_HD_MAX     EQU     .2              ; Maximum value for high digit of hour
00000003        00081 OPTION_SETUP    EQU     b'00000011'     ; TMR0 - internal, /16 prescale
00000007        00082 BUZINITVAL      EQU     7               ;
00000008        00083 INIT_MODE_COUNT EQU     8               ; Digit counts to move to hour digits
00000028        00084 ALARMCYCCNT     EQU     .40             ; Alarm for 10 seconds (ALARMCYCCNT/4)
                00085
01FF            00086         ORG     01FFh                   ; The PICSX reset vector is at end of memory
01FF            00087 reset_vector
01FF 0BA8       00088         GOTO    init                    ; Jump to the initialization code
                00089
0000            00090         ORG     0
                00091 ; ************************************
                00092 ; * Current mode look-up table       *
                00093 ; ************************************
0000            00094 mode_timer
0000 0E03       00095         ANDLW   3                       ; Mask off upper bits just in case
0001 01E2       00096         ADDWF   PCL,F                   ; Jump to one of 4 look-up entries
0002 0814       00097         RETLW   TMR_SEC_LD              ; Return the address of the 99 min timer RAM
0003 0810       00098         RETLW   ALM_MIN_LD              ; Return the address of the alarm RAM
0004 080C       00099         RETLW   CLK_MIN_LD              ; Return the address of the clock RAM
0005 080C       00100         RETLW   CLK_MIN_LD              ; Return the address of the clock RAM
                00101
                00102 ; ************************************
                00103 ; * Buzz the buzzer for 1/8 second    *
                00104 ; ************************************
0006            00105 buzz_now
0006 0066       00106         CLRF    PORTB                   ; Shut off the segments
0007            00107 buzz_now_dispon
0007 007C       00108         CLRF    TEMP                    ; Buzz for 256 pulses
0008            00109 loop_buz
0008 05E6       00110         BSF     BUZZEROUT               ; Send out pulse
0009 04E6       00111         BCF     BUZZEROUT               ; Clear out the pulse
000A 02FC       00112         DECFSZ  TEMP,F                  ; Decrement counter and skip when done
000B 0A08       00113         GOTO    loop_buz                ; Go back and send another pulse
000C 0800       00114         RETLW   0                       ; We are done so come back!
                00115
                00116 ; ************************************
                00117 ; * Mux drive the next LED display digit *
                00118 ; ************************************
```


© 1997 Microchip Technology Inc.

```
000D            00119 task_scan   ; (19 (next_scan) + 2 = 21 cycles - must be called every 11 cy)
000D 0601       00120         BTFSC   SCANBIT         ; Synch up with 3.9 mS timer bit
000E 0A0D       00121         GOTO    task_scan       ; Jump back until bit is clear
                00122
000F            00123 next_scan   ; (15 + 2 call + 2 return = 19 cycles)
000F 035B       00124         RLF     PREVSCAN,W      ; Move to the next digit select into C
0010 073B       00125         BTFSS   PREVSCAN,1      ; 0 Check if display A was on before
0011 0209       00126         MOVF    DISPSEGS_C,W    ; Place display B value into W
0012 071B       00127         BTFSS   PREVSCAN,0      ; 1 Check if display B was on before
0013 0208       00128         MOVF    DISPSEGS_B,W    ; Place display C value into W
0014 077B       00129         BTFSS   PREVSCAN,3      ; 2 Check if display C was on before
0015 0207       00130         MOVF    DISPSEGS_A,W    ; Place display D value into W
0016 075B       00131         BTFSS   PREVSCAN,2      ; 3 Check if display D was on before
0017 020A       00132         MOVF    DISPSEGS_D,W    ; Place display A value into W
0018 0066       00133         CLRF    PORTB           ; Turn off all segments
0019 037B       00134         RLF     PREVSCAN,F      ; Move to the next digit
001A 0365       00135         RLF     PORTA,F         ; Move port to the next digit
001B 0026       00136         MOVWF   PORTB           ; Place next segment value on PORTB
001C 021B       00137         MOVF    PREVSCAN,W      ; Restore the port in case it is wrong
001D 0025       00138         MOVWF   PORTA           ; Restore the port
001E 0800       00139         RETLW   0               ; Display is updated - now return
                00140
                00141
                00142 ; *********************************************
                00143 ; * Move new digit display info out to display *
                00144 ; *********************************************
001F            00145 disp_value
001F 0024       00146         MOVWF   FSR             ; Place W into FSR for indirect addressing
0020 090D       00147         CALL    task_scan       ; Scan the next LED digit.
0021 0200       00148         MOVF    INDADDR,W       ; Place display value into W
0022 0937       00149         CALL    led_lookup      ; Look up seven segment value
0023 0027       00150         MOVWF   DISPSEGS_A      ; Move value out to display register A
0024 02A4       00151         INCF    FSR,F           ; Go to next display value
0025 090D       00152         CALL    task_scan       ; Scan the next LED digit.
0026 0200       00153         MOVF    INDADDR,W       ; Place display value into W
0027 0937       00154         CALL    led_lookup      ; Look up seven segment value
0028 0028       00155         MOVWF   DISPSEGS_B      ; Move value out to display register B
0029 02A4       00156         INCF    FSR,F           ; Go to next display value
002A 090D       00157         CALL    task_scan       ; Scan the next LED digit.
002B 0200       00158         MOVF    INDADDR,W       ; Place display value into W
002C 0937       00159         CALL    led_lookup      ; Look up seven segment value
002D 0029       00160         MOVWF   DISPSEGS_C      ; Move value out to display register C
002E 02A4       00161         INCF    FSR,F           ; Go to next display value
002F 090D       00162         CALL    task_scan       ; Scan the next LED digit.
0030 0200       00163         MOVF    INDADDR,W       ; Place display value into W
0031 0643       00164         BTFSC   STATUS,Z        ; ZBLANK - Check for a zero
0032 0240       00165         COMF    INDADDR,W       ; ZBLANK - Clear digit with FF if leading 0
0033 0937       00166         CALL    led_lookup      ; Look up seven segment value
0034 002A       00167         MOVWF   DISPSEGS_D      ; Move value out to display register D
0035 090D       00168         CALL    task_scan       ; Scan the next LED digit.
0036 0800       00169         RETLW   0
                00170
                00171 ; ***************************************
                00172 ; * Convert display value into segments  *
                00173 ; ***************************************
0037            00174 led_lookup
0037 0E0F       00175         ANDLW   0Fh             ; Strip off upper digits
0038 01E2       00176         ADDWF   PCL,F           ; Jump into the correct location
0039 083F       00177         RETLW   b'00111111'     ; Bit pattern for a Zero
003A 0806       00178         RETLW   b'00000110'     ; Bit pattern for a One
003B 085B       00179         RETLW   b'01011011'     ; Bit pattern for a Two
003C 084F       00180         RETLW   b'01001111'     ; Bit pattern for a Three
003D 0866       00181         RETLW   b'01100110'     ; Bit pattern for a Four
003E 086D       00182         RETLW   b'01101101'     ; Bit pattern for a Five
003F 087D       00183         RETLW   b'01111101'     ; Bit pattern for a Six
0040 0807       00184         RETLW   b'00000111'     ; Bit pattern for a Seven
```


© 1997 Microchip Technology Inc.

```
0041 087F      00185          RETLW   b'01111111'     ; Bit pattern for a Eight
0042 086F      00186          RETLW   b'01101111'     ; Bit pattern for a Nine
0043 0800      00187          RETLW   0               ; Turn display off - ILLEGAL VALUE
0044 0800      00188          RETLW   0               ; Turn display off - ILLEGAL VALUE
0045 0800      00189          RETLW   0               ; Turn display off - ILLEGAL VALUE
0046 0800      00190          RETLW   0               ; Turn display off - ILLEGAL VALUE
0047 0800      00191          RETLW   0               ; Turn display off - ILLEGAL VALUE
0048 0800      00192          RETLW   0               ; Turn display off - ILLEGAL VALUE
               00193
               00194 ; ************************************************************************
               00195 ; * Convert display value into single segment ON for manufacturing diags *
               00196 ; ************************************************************************
0049           00197 mfg_led_lookup
0049 0E07      00198          ANDLW   07h             ; Strip off upper digits
004A 01E2      00199          ADDWF   PCL,F           ; Jump into the correct location
004B 0801      00200          RETLW   b'00000001'     ; Bit pattern for segment A on only
004C 0802      00201          RETLW   b'00000010'     ; Bit pattern for segment B on only
004D 0804      00202          RETLW   b'00000100'     ; Bit pattern for segment C on only
004E 0808      00203          RETLW   b'00001000'     ; Bit pattern for segment D on only
004F 0810      00204          RETLW   b'00010000'     ; Bit pattern for segment E on only
0050 0820      00205          RETLW   b'00100000'     ; Bit pattern for segment F on only
0051 0840      00206          RETLW   b'01000000'     ; Bit pattern for segment G on only
0052 087F      00207          RETLW   b'01111111'     ; Bit pattern for all segments on
               00208
               00209 ; ********************************************************
               00210 ; * Wake-up and turn on the displays                      *
               00211 ; ********************************************************
0053           00212 turnon_scan
0053 059D      00213          BSF     DISPON          ; Set display ON bit
0054 0CEE      00214          MOVLW   b'11101110'     ; Place digit 0 scan pattern in W
0055 019B      00215          XORWF   PREVSCAN,W      ; See if this is the current scan
0056 0643      00216          BTFSC   STATUS,Z        ; Skip if this is not the current scan
0057 0800      00217          RETLW   0               ; Legal scan value - we are done!
0058 0CDD      00218          MOVLW   b'11011101'     ; Place digit 1 scan pattern in W
0059 019B      00219          XORWF   PREVSCAN,W      ; See if this is the current scan
005A 0643      00220          BTFSC   STATUS,Z        ; Skip if this is not the current scan
005B 0800      00221          RETLW   0               ; Legal scan value - we are done!
005C 0CBB      00222          MOVLW   b'10111011'     ; Place digit 2 scan pattern in W
005D 019B      00223          XORWF   PREVSCAN,W      ; See if this is the current scan
005E 0643      00224          BTFSC   STATUS,Z        ; Skip if this is not the current scan
005F 0800      00225          RETLW   0               ; Legal scan value - we are done!
0060 0C77      00226          MOVLW   b'01110111'     ; Place digit 3 scan pattern in W
0061 019B      00227          XORWF   PREVSCAN,W      ; See if this is the current scan
0062 0643      00228          BTFSC   STATUS,Z        ; Skip if this is not the current scan
0063 0800      00229          RETLW   0               ; Legal scan value - we are done!
0064 0CEE      00230          MOVLW   0EEh            ; Move digit 0 scan value into W
0065 003B      00231          MOVWF   PREVSCAN        ; Move it into scan pattern register
               00232
               00233 ; ***************************************
               00234 ; * Scan for pressed keys                *
               00235 ; ***************************************
0066           00236 scan_keys
0066 0066      00237          CLRF    PORTB           ; Turn off all of the segments
0067 0CFF      00238          MOVLW   0FFh            ; Place FF into W
0068 0025      00239          MOVWF   PORTA           ; Make PORT A all ones
0069 0C70      00240          MOVLW   b'01110000'     ; Place 70 into W
006A 0006      00241          TRIS    PORTB           ; Make RB4,5,6 inputs others outputs
006B 0206      00242          MOVF    PORTB,W         ; Place keyscan value into W
006C 0198      00243          XORWF   KEYPAT,W        ; Place Delta key press into W
006D 003C      00244          MOVWF   TEMP            ; Place Delta key press into TEMP
006E 01B8      00245          XORWF   KEYPAT,F        ; Update KEYPAT reg to buttons pressed
006F 0040      00246          CLRW                    ; Place 0 into W
0070 0006      00247          TRIS    PORTB           ; Make PORT B outputs
0071 021B      00248          MOVF    PREVSCAN,W      ; Place previous scan value into W
0072 0025      00249          MOVWF   PORTA           ; Turn on the scan
0073 0800      00250          RETLW   0
```


© 1997 Microchip Technology Inc.

```
                        00251 ; ***************************************
                        00252 ; * Check if alarm or timer is expired   *
                        00253 ; ***************************************
0074                    00254 check_time
0074 090D               00255         CALL    task_scan       ; Scan the next LED digit.
0075 0579               00256         BSF     ALARMNOW        ; Set the alarm bit
0076 0599               00257         BSF     EGGNOW          ; Set the Egg timer alarm bit
0077 0210               00258         MOVF    ALM_MIN_LD,W    ; Place alarm minute counter into W
0078 008C               00259         SUBWF   CLK_MIN_LD,W    ; CLK_MIN_LD - W -> W
0079 0743               00260         BTFSS   STATUS,Z        ; Skip if they are equal
007A 0479               00261         BCF     ALARMNOW        ; They are not equal so clear alarm bit
007B 0211               00262         MOVF    ALM_MIN_HD,W    ; Place alarm minute counter into W
007C 008D               00263         SUBWF   CLK_MIN_HD,W    ; CLK_MIN_HD - W -> W
007D 0743               00264         BTFSS   STATUS,Z        ; Skip if they are equal
007E 0479               00265         BCF     ALARMNOW        ; They are not equal so clear alarm bit
007F 090D               00266         CALL    task_scan       ; Scan the next LED digit.
0080 0212               00267         MOVF    ALM_HOUR_LD,W   ; Place alarm hour counter into W
0081 008E               00268         SUBWF   CLK_HOUR_LD,W   ; CLK_HOUR_LD - W -> W
0082 0743               00269         BTFSS   STATUS,Z        ; Skip if they are equal
0083 0479               00270         BCF     ALARMNOW        ; They are not equal so clear alarm bit
0084 0213               00271         MOVF    ALM_HOUR_HD,W   ; Place alarm hour counter into W
0085 008F               00272         SUBWF   CLK_HOUR_HD,W   ; CLK_HOUR_LD - W -> W
0086 0743               00273         BTFSS   STATUS,Z        ; Skip if they are equal
0087 0479               00274         BCF     ALARMNOW        ; They are not equal so clear alarm bit
0088 090D               00275         CALL    task_scan       ; Scan the next LED digit.
0089 0214               00276         MOVF    TMR_SEC_LD,W    ; Set the Z bit to check for zero
008A 0743               00277         BTFSS   STATUS,Z        ; Skip if this digit is zero
008B 0499               00278         BCF     EGGNOW          ; Timer is not zero so clear egg alarm bit
008C 0215               00279         MOVF    TMR_SEC_HD,W    ; Set the Z bit to check for zero
008D 0743               00280         BTFSS   STATUS,Z        ; Skip if this digit is zero
008E 0499               00281         BCF     EGGNOW          ; Timer is not zero so clear egg alarm bit
008F 0216               00282         MOVF    TMR_MIN_LD,W    ; Set the Z bit to check for zero
0090 0743               00283         BTFSS   STATUS,Z        ; Skip if this digit is zero
0091 0499               00284         BCF     EGGNOW          ; Timer is not zero so clear egg alarm bit
0092 090D               00285         CALL    task_scan       ; Scan the next LED digit.
0093 0217               00286         MOVF    TMR_MIN_HD,W    ; Set the Z bit to check for zero
0094 0743               00287         BTFSS   STATUS,Z        ; Skip if this digit is zero
0095 0499               00288         BCF     EGGNOW          ; Timer is not zero so clear egg alarm bit
0096 0799               00289         BTFSS   EGGNOW          ; Skip if we are still at EGG Time
0097 05C3               00290         BSF     EGGOK           ; If we are not at EGG time, re-set egg alarm
0098 0779               00291         BTFSS   ALARMNOW        ; Skip if we are still at Alarm time
0099 05A3               00292         BSF     ALARMOK         ; If we are not at Alarm time, re-set alarm
009A 090D               00293         CALL    task_scan       ; Scan the next LED digit.
009B 0800               00294         RETLW   0
                        00295
                        00296 ; ***************************************
                        00297 ; * Increment the clock, timer or alarm  *
                        00298 ; ***************************************
009C                    00299 inc_time
009C 0024               00300         MOVWF   FSR             ; Add one to clock second counter
009D 090D               00301         CALL    task_scan       ; Scan the next LED digit.
009E 02A0               00302         INCF    INDADDR,f       ; Add one to minute lower digit
009F 0C3C               00303         MOVLW   SEC_MAX         ; Place second max value into w
00A0 0080               00304         SUBWF   INDADDR,W       ; CLOCK_SEC - SEC_MAX -> W
00A1 0703               00305         BTFSS   STATUS,C        ; Skip if there is an overflow
00A2 0800               00306         RETLW   0               ; We are done so let's get out of here!
00A3 006B               00307         CLRF    CLK_SEC         ; Clear CLK_second counter
00A4 02A4               00308         INCF    FSR,F           ; Move to the next digit
00A5 02A0               00309         INCF    INDADDR,F       ; Add 1 to minute LOW digit
00A6 0AA9               00310         GOTO    skip_min_fsr    ; Jump to the next digit
00A7                    00311 inc_min_ld
00A7 0024               00312         MOVWF   FSR
00A8 02A0               00313         INCF    INDADDR,F       ; Add 1 to minute LOW digit
00A9                    00314 skip_min_fsr
00A9 090D               00315         CALL    task_scan       ; Scan the next LED digit.
00AA 0C0A               00316         MOVLW   MIN_LD_MAX      ; Place minute lower digit max value into W
```

5

© 1997 Microchip Technology Inc.

```
00AB 0080      00317          SUBWF   INDADDR,W       ; CLK_MIN_LD - MIN_LD_MAX -> W
00AC 0703      00318          BTFSS   STATUS,C        ; Skip if there is an overflow
00AD 0800      00319          RETLW   0               ; We are done so let's get out of here!
00AE 0060      00320          CLRF    INDADDR         ; Clear CLK minute low digit
00AF 02A4      00321          INCF    FSR,F           ; Move to the minute high digit
00B0 02A0      00322          INCF    INDADDR,F       ; Add one to minute high digit
00B1           00323 inc_min_hd
00B1 090D      00324          CALL    task_scan       ; Scan the next LED digit.
00B2 0C06      00325          MOVLW   MIN_HD_MAX      ; Place minute high digit max value into W
00B3 0080      00326          SUBWF   INDADDR,W       ; CLK_MIN_HD - MIN_HD_MAX -> W
00B4 0703      00327          BTFSS   STATUS,C        ; Skip if there is an overflow
00B5 0800      00328          RETLW   0               ; We are done so let's get out of here!
00B6 0060      00329          CLRF    INDADDR         ; Clear CLK minute high digit
00B7 02A4      00330          INCF    FSR,F           ; Move to the hour low digit
00B8 02A0      00331          INCF    INDADDR,F       ; Add one to hour low digit
00B9 0ABE      00332          GOTO    skip_hour_fsr   ; Jump to the next digit
00BA           00333 inc_hour_ld
00BA 0024      00334          MOVWF   FSR
00BB 02A4      00335          INCF    FSR,F
00BC 02A4      00336          INCF    FSR,F
00BD 02A0      00337          INCF    INDADDR,F       ; Add 1 to minute LOW digit
00BE           00338 skip_hour_fsr
00BE 090D      00339          CALL    task_scan       ; Scan the next LED digit.
00BF 0C0A      00340          MOVLW   MIN_LD_MAX      ; Place hour lower digit max value into W
00C0 0080      00341          SUBWF   INDADDR,W       ; CLK_HOUR_LD - HOUR_LD_MAX -> W
00C1 0703      00342          BTFSS   STATUS,C        ; Skip if there is an overflow
00C2 0AC7      00343          GOTO    check_inc       ; We need to check for overflow
00C3 0060      00344          CLRF    INDADDR         ; Clear CLK hour low digit
00C4 02A4      00345          INCF    FSR,F           ; Move to the hour high digit
00C5 02A0      00346          INCF    INDADDR,F       ; Add one to hour high digit
00C6 0AC8      00347          GOTO    inc_hour_hd
00C7           00348 check_inc
00C7 02A4      00349          INCF    FSR,F           ; Move to hour high digit
00C8           00350 inc_hour_hd
00C8 090D      00351          CALL    task_scan       ; Scan the next LED digit.
00C9 0C02      00352          MOVLW   HOUR_HD_MAX     ; Place hour high digit max value into W
00CA 0639      00353          BTFSC   FLAGS,1
00CB 0ACE      00354          GOTO    off_mode1
00CC 0619      00355          BTFSC   FLAGS,0
00CD 0C09      00356          MOVLW   MIN_LD_MAX-1
00CE           00357 off_mode1
00CE 0080      00358          SUBWF   INDADDR,W       ; CLK_HOUR_HD - HOUR_HD_MAX -> W
00CF 0703      00359          BTFSS   STATUS,C        ; Skip if there is an overflow
00D0 0800      00360          RETLW   0               ; We are done so let's get out of here!
00D1 00E4      00361          DECF    FSR,F           ; Move to the hour low digit
00D2 090D      00362          CALL    task_scan       ; Scan the next LED digit.
00D3 0C04      00363          MOVLW   HOUR_LD_MAX     ; Place hour high digit max value into W
00D4 0639      00364          BTFSC   FLAGS,1
00D5 0AD8      00365          GOTO    off_mode2
00D6 0619      00366          BTFSC   FLAGS,0
00D7 0C00      00367          MOVLW   0               ; Clear W
00D8           00368 off_mode2
00D8 0080      00369          SUBWF   INDADDR,W       ; CLK_HOUR_HD - HOUR_HD_MAX -> W
00D9 0703      00370          BTFSS   STATUS,C        ; Skip if there is an overflow
00DA 0800      00371          RETLW   0               ; We are done so let's get out of here!
00DB 090D      00372          CALL    task_scan       ; Scan the next LED digit.
00DC 0060      00373          CLRF    INDADDR         ; Clear hour high digit
00DD 0639      00374          BTFSC   FLAGS,1
00DE 0AE0      00375          GOTO    off_mode3
00DF 0719      00376          BTFSS   FLAGS,0
00E0           00377 off_mode3
00E0 0000      00378          NOP
00E1 02A4      00379          INCF    FSR,F           ; Move to the hour high digit
00E2 0060      00380          CLRF    INDADDR         ; Clear one hour low digit
00E3 090D      00381          CALL    task_scan
00E4 0800      00382          RETLW   0               ; We are done so let's get out of here!
```


© 1997 Microchip Technology Inc.

```
                00383
00E5            00384 dec_hour_ld
00E5 0AF9       00385         GOTO    dec_hour_ld_vect  ; ran out of CALL space....
                00386
                00387 ; ***************************************
                00388 ; * Decrement the clock, alarm or timer  *
                00389 ; ***************************************
00E6            00390 dec_time
00E6            00391 dec_min_ld
00E6 0024       00392         MOVWF   FSR               ; Set up pointer for indirect address
00E7 090D       00393         CALL    task_scan         ; Scan the next LED digit.
00E8 00E0       00394         DECF    INDADDR,F         ; Subtract one from CLK_MIN_LD
00E9 0240       00395         COMF    INDADDR,W         ; Set the Z bit to check for zero
00EA 0743       00396         BTFSS   STATUS,Z          ; Skip if CLK_MIN_LD is zero
00EB 0800       00397         RETLW   0                 ; We are done... Let's get out of here
00EC 0C09       00398         MOVLW   MIN_LD_MAX - 1    ; Place minute lower digit max value into W
00ED 0020       00399         MOVWF   INDADDR           ; MIN_LD_MAX -> CLK_MIN_LD
00EE            00400 dec_min_hd
00EE 090D       00401         CALL    task_scan         ; Scan the next LED digit.
00EF 02A4       00402         INCF    FSR,F             ; Move the pointer to Min HIGH DIGIT
00F0 00E0       00403         DECF    INDADDR,F         ; Subtract one from CLK_MIN_HD
00F1 0240       00404         COMF    INDADDR,W         ; Set the Z bit to check for zero
00F2 0743       00405         BTFSS   STATUS,Z          ; Skip if CLK_MIN_LD is zero
00F3 0800       00406         RETLW   0                 ; We are done... Let's get out of here
00F4 0C05       00407         MOVLW   MIN_HD_MAX - 1    ; Place minute lower digit max value into W
00F5 0020       00408         MOVWF   INDADDR           ; MIN_HD_MAX -> CLK_MIN_HD
00F6 090D       00409         CALL    task_scan         ; Scan the next LED digit.
00F7 02A4       00410         INCF    FSR,F             ; Move the pointer to Hour LOW DIGIT
00F8 0AFD       00411         GOTO    skip_dhour_fsr    ; Jump to the next digit
00F9            00412 dec_hour_ld_vect
00F9 0024       00413         MOVWF   FSR
00FA 02A4       00414         INCF    FSR,F
00FB 02A4       00415         INCF    FSR,F
00FC 090D       00416         CALL    task_scan         ; Scan the next LED digit.
00FD            00417 skip_dhour_fsr
00FD 00E0       00418         DECF    INDADDR,F         ; Subtract one from CLK_HOUR_LD
00FE 0240       00419         COMF    INDADDR,W         ; Set the Z bit to check for zero
00FF 0743       00420         BTFSS   STATUS,Z          ; Skip if CLK_MIN_LD is zero
0100 0B06       00421         GOTO    check_hour
0101 0C09       00422         MOVLW   MIN_LD_MAX - 1    ; Place minute lower digit max value into W
0102 0020       00423         MOVWF   INDADDR           ; MIN_LD_MAX -> CLK_HOUR_LD
0103 02A4       00424         INCF    FSR,F             ; Move the pointer to Hour HIGH DIGIT
0104 00E0       00425         DECF    INDADDR,F         ; Subtract one from CLK_HOUR_HD
0105 0B07       00426         GOTO    dec_hour_hd
0106            00427 check_hour
0106 02A4       00428         INCF    FSR,F             ; Point to hour high digit
0107            00429 dec_hour_hd
0107 090D       00430         CALL    task_scan         ; Scan the next LED digit.
0108 0240       00431         COMF    INDADDR,W
0109 0743       00432         BTFSS   STATUS,Z
010A 0800       00433         RETLW   0
010B 090D       00434         CALL    task_scan         ; Scan the next LED digit.
010C 00E4       00435         DECF    FSR,F
010D 0C09       00436         MOVLW   .9                ; Reset digit to 9
010E 0080       00437         SUBWF   INDADDR,W
010F 0743       00438         BTFSS   STATUS,Z          ; Skip if CLK_MIN_LD is zero
0110 0800       00439         RETLW   0                 ; We are done... Let's get out of here
0111 090D       00440         CALL    task_scan         ; Scan the next LED digit.
0112 02A4       00441         INCF    FSR,F
0113 0C02       00442         MOVLW   HOUR_HD_MAX       ; Place minute lower digit max value into W
0114 0739       00443         BTFSS   FLAGS,1           ; Skip if CLOCK or ALARM mode
0115 0C09       00444         MOVLW   .9                ; Reset digit to 9
0116 0020       00445         MOVWF   INDADDR           ; HOUR_HD_MAX -> CLK_HOUR_HD
0117 0C03       00446         MOVLW   HOUR_LD_MAX - 1   ; Place minute lower digit max value into W
0118 0739       00447         BTFSS   FLAGS,1           ; Skip if CLOCK or ALARM mode
0119 0C09       00448         MOVLW   .9                ; Reset digit to 9
```

5

```
011A 00E4      00449          DECF    FSR,F           ; Move the pointer to Min LOW DIGIT
011B 0020      00450          MOVWF   INDADDR         ; HOUR_LD_MAX -> CLK_HOUR_LD
011C 090D      00451          CALL    task_scan       ; Scan the next LED digit.
011D 0800      00452          RETLW   0               ; We are done... Let's get out of here
               00453
               00454 ; ************************************
               00455 ; * Main loop calls all tasks as needed  *
               00456 ; ************************************
011E           00457 main_loop
011E 090D      00458          CALL    task_scan       ; Scan the next LED digit.
011F 0201      00459          MOVF    TMR0,W          ; Place current TMR0 value into W
0120 019A      00460          XORWF   PREVTMR0,W      ; Lets see which bits have changed...
0121 003C      00461          MOVWF   TEMP            ; All changed bits are placed in temp for test
0122 01BA      00462          XORWF   PREVTMR0,F      ; Update Previous TMR0 value.
0123 07FC      00463          BTFSS   SECBIT          ; Skip if it is not time to increment second
0124 0B1E      00464          GOTO    main_loop       ; Go back to main loop if 250 mS not passed
0125 0C20      00465          MOVLW   b'00100000'     ; Bits 6 and 5 of FLAGS used as divide by 4
0126 01F9      00466          ADDWF   FLAGS,F         ; Add one to bit 5
0127 07F9      00467          BTFSS   TIMENOW         ; Check bit 7 - if four adds occur, skip
0128 0B38      00468          GOTO    skip_timer      ; One second has not passed - skip timers
0129 090D      00469          CALL    task_scan       ; Scan the next LED digit.
012A 04F9      00470          BCF     TIMENOW         ; Clear out second passed flag
012B 0C0B      00471          MOVLW   CLK_SEC         ; Place pointer to increment clock
012C 099C      00472          CALL    inc_time        ; Increment the clock
012D 0974      00473          CALL    check_time      ; Check for alarm or timer conditions
012E 0699      00474          BTFSC   EGGNOW          ; Do NOT decrease timer if zero
012F 0B38      00475          GOTO    skip_timer      ; Jump out if egg timer is zero
0130 06D8      00476          BTFSC   UPKEY           ; Skip if UP key is NOT pressed
0131 0B38      00477          GOTO    skip_timer      ; Jump out if UP key is pressed
0132 06B8      00478          BTFSC   DOWNKEY         ; Skip if DOWN key is NOT pressed
0133 0B38      00479          GOTO    skip_timer      ; Jump out if DOWN key is pressed
0134 0C14      00480          MOVLW   TMR_SEC_LD      ; Place pointer to decrement timer
0135 09E6      00481          CALL    dec_time        ; Decrement countdown timer
0136 0C28      00482          MOVLW   ALARMCYCCNT     ; Place the number of alarm beeps into W
0137 003F      00483          MOVWF   ALARMCNT        ; Move beep count to ALARMCNT
0138           00484 skip_timer
0138 07A3      00485          BTFSS   ALARMOK         ; Skip if this is the first pass into alarm
0139 0B3F      00486          GOTO    skip_wakeup     ; Second pass - do not re-init ALARMCNT
013A 0779      00487          BTFSS   ALARMNOW        ; Skip if this is alarm pass
013B 0B3F      00488          GOTO    skip_wakeup     ; Countdown timer - do not re-init ALARMCNT
013C 0C28      00489          MOVLW   ALARMCYCCNT     ; Place the number of alarm beeps into W
013D 003F      00490          MOVWF   ALARMCNT        ; Move beep count to ALARMCNT
013E 04A3      00491          BCF     ALARMOK         ; Clear flag for second pass
013F           00492 skip_wakeup
013F 090D      00493          CALL    task_scan       ; Scan the next LED digit.
0140 0679      00494          BTFSC   ALARMNOW        ; Skip if alarm clock is not set
0141 0B45      00495          GOTO    send_alarm      ; Blast out a beep
0142 0699      00496          BTFSC   EGGNOW          ; Skip if countdown timer is not alarming
0143 0B45      00497          GOTO    send_alarm      ; Blast out a beep
0144 0B4A      00498          GOTO    skip_alarm      ; Skip beeping and continue
0145           00499 send_alarm
0145 021F      00500          MOVF    ALARMCNT,W      ; Place ALARMCNT into W
0146 0643      00501          BTFSC   STATUS,Z        ; Skip if not zero
0147 0B4A      00502          GOTO    skip_alarm      ; We are done beeping - skip and continue
0148 02FF      00503          DECFSZ  ALARMCNT,F      ; Decrement beep count and skip when zero
0149 0906      00504          CALL    buzz_now        ; Blast out the beep!!!
014A           00505 skip_alarm
014A 07B9      00506          BTFSS   FLAGS,5         ; Skip if it is time to scan the keys 1/2 sec
014B 0B9A      00507          goto    finish_update   ; Jump to finish updates - don't scan
014C 0966      00508          CALL    scan_keys       ; Scan the keys and load value into KEYPAT
014D 090D      00509          CALL    task_scan       ; Scan the next LED digit.
014E 0798      00510          BTFSS   MODEKEY         ; Skip if the MODEKEY is pressed
014F 0B55      00511          GOTO    same_mode       ; Not pressed so it is the same mode...
0150 079C      00512          BTFSS   MODEKEYCHG      ; Skip if the is pressing edge
0151 0B55      00513          GOTO    same_mode       ; Button is held so it is the same mode...
0152 02B9      00514          INCF    FLAGS,F         ; Advance the mode by incrimenting bits 0,1
```


© 1997 Microchip Technology Inc.

```
0153 0459      00515          BCF     FLAGS,2          ; Force mode to wrap-around by clearing bit 2
0154 0953      00516          CALL    turnon_scan      ; Mode button pressed - must turn on LEDs
               00517
0155           00518 same_mode
0155 090D      00519          call    task_scan        ; Scan the next LED digit.
0156 06D8      00520          BTFSC   UPKEY            ; Skip if the UP key is not pressed
0157 0B66      00521          GOTO    serve_up_key     ; UP key is pressed - jump to serve it!
0158 06B8      00522          BTFSC   DOWNKEY          ; Skip if the DOWN key is not pressed
0159 0B81      00523          GOTO    serve_down_key   ; DOWN key is pressed - jump to serve it!
015A 0C08      00524          MOVLW   INIT_MODE_COUNT  ; UP and DOWN not pressed - re-init mode count
015B 003E      00525          MOVWF   MODE_COUNT       ; Change back to lower digits for setting
015C 023D      00526          MOVF    DISPONCNT,F      ; Update Z bit in STATUS reg display on time
015D 0743      00527          BTFSS   STATUS,Z         ; Skip if displays should be OFF
015E 00FD      00528          DECF    DISPONCNT,F      ; Decrement display ON counter
015F 0743      00529          BTFSS   STATUS,Z         ; Skip if displays should be OFF
0160 0B9A      00530          GOTO    finish_update    ; Displays are ON - jump to finish updates
0161 0419      00531          BCF     FLAGS,0          ; Restore the mode to displays OFF
0162 0439      00532          BCF     FLAGS,1          ; Restore the mode to displays OFF
0163 0066      00533          CLRF    PORTB            ; Clear out segment drives on PORTB
0164 0065      00534          CLRF    PORTA            ; Clear out common digit drives on PORTA
0165 0B9A      00535          GOTO    finish_update    ; Jump to finish updates
0166           00536 serve_up_key
0166 090D      00537          call    task_scan        ; Scan the next LED digit.
0167 0619      00538          BTFSC   FLAGS,0          ; Skip if not in TIMER or CLOCK mode
0168 0B6D      00539          GOTO    no_up_display    ; Currently in TIMER or CLOCK - keep mode
0169 0639      00540          BTFSC   FLAGS,1          ; Skip if not in ALARM mode
016A 0B6D      00541          GOTO    no_up_display    ; Currently in ALARM - keep mode
016B 0519      00542          BSF     FLAGS,0          ; Set to CLOCK mode
016C 0539      00543          BSF     FLAGS,1          ; Set to CLOCK mode
016D           00544 no_up_display
016D 007F      00545          CLRF    ALARMCNT         ; A key was pressed, so turn off alarm
016E 0953      00546          call    turnon_scan      ; Turn on the LEDs
016F 0798      00547          BTFSS   MODEKEY          ; Skip if MODE is pressed as well
0170 0B9A      00548          GOTO    finish_update    ; MODE is not pressed - jump to finish update
0171 021E      00549          MOVF    MODE_COUNT,W     ; Update STATUS Z bit for mode count
0172 0743      00550          BTFSS   STATUS,Z         ; Skip if we have counted down to zero
0173 00FE      00551          DECF    MODE_COUNT,F     ; Decrement the mode count
0174 090D      00552          call    task_scan        ; Scan the next LED digit.
0175 021E      00553          MOVF    MODE_COUNT,W     ; Update the Z bit to check for zero
0176 0743      00554          BTFSS   STATUS,Z         ; Skip if we have incrimented for 7 times
0177 0B7C      00555          GOTO    serve_min_up     ; Increment the minutes digits
0178 00D9      00556          DECF    FLAGS,W          ; Place current mode into W
0179 0900      00557          CALL    mode_timer       ; Look-up register RAM address for current mode
017A 09BA      00558          CALL    inc_hour_ld      ; Add one hour to the current display
017B 0B9A      00559          GOTO    finish_update    ; Jump to finish updates
017C           00560 serve_min_up
017C 090D      00561          call    task_scan        ; Scan the next LED digit.
017D 00D9      00562          DECF    FLAGS,W          ; Place current mode into W
017E 0900      00563          CALL    mode_timer       ; Look-up register RAM address for current mode
017F 09A7      00564          CALL    inc_min_ld       ; Add one minute to the current display
0180 0B9A      00565          GOTO    finish_update    ; Jump to finish updates
0181           00566 serve_down_key
0181 090D      00567          call    task_scan        ; Scan the next LED digit.
0182 0619      00568          BTFSC   FLAGS,0          ; Skip if not in TIMER or CLOCK mode
0183 0B88      00569          GOTO    no_dn_display    ; Currently in TIMER or CLOCK - keep mode
0184 0639      00570          BTFSC   FLAGS,1          ; Skip if not in ALARM mode
0185 0B88      00571          GOTO    no_dn_display    ; Currently in ALARM - keep mode
0186 0519      00572          BSF     FLAGS,0          ; Set to CLOCK mode
0187 0539      00573          BSF     FLAGS,1          ; Set to CLOCK mode
0188           00574 no_dn_display
0188 007F      00575          CLRF    ALARMCNT         ; A key was pressed, so turn off alarm
0189 0953      00576          CALL    turnon_scan      ; Turn on the LEDs
018A 0798      00577          BTFSS   MODEKEY          ; Skip if MODE is pressed as well
018B 0B9A      00578          GOTO    finish_update    ; MODE is not pressed - jump to finish update
018C 021E      00579          MOVF    MODE_COUNT,W     ; Update STATUS Z bit for mode count
018D 0743      00580          BTFSS   STATUS,Z         ; Skip if we have counted down to zero
```


© 1997 Microchip Technology Inc.

```
018E 00FE      00581          DECF    MODE_COUNT,F    ; Decrement the mode count
               00582
018F 090D      00583          call    task_scan       ; Scan the next LED digit.
0190 021E      00584          MOVF    MODE_COUNT,W    ; Update the Z bit to check for zero
0191 0743      00585          BTFSS   STATUS,Z        ; Skip if we have incrimented for 7 times
0192 0B97      00586          GOTO    serve_min_down  ; Decrement the minutes digits
0193 00D9      00587          DECF    FLAGS,W         ; Place current mode into W
0194 0900      00588          CALL    mode_timer      ; Look-up register RAM address for current mode
0195 09E5      00589          CALL    dec_hour_ld     ; Subtract one hour from the current display
0196 0B9A      00590          GOTO    finish_update   ; Jump to finish updates
0197           00591 serve_min_down
0197 00D9      00592          DECF    FLAGS,W         ; Place current mode into W
0198 0900      00593          CALL    mode_timer      ; Look-up register RAM address for current mode
0199 09E6      00594          CALL    dec_min_ld      ; Subtract one minute from the current display
019A           00595 finish_update
019A 090D      00596          call    task_scan       ; Scan the next LED digit.
019B 0619      00597          BTFSC   FLAGS,0         ; Skip if in mode OFF or ALARM
019C 0BA4      00598          GOTO    new_display     ; Jump to update LED display registers
019D 0639      00599          BTFSC   FLAGS,1         ; Skip if in mode OFF
019E 0BA4      00600          GOTO    new_display     ; Jump to update LED display registers
019F 0067      00601          CLRF    DISPSEGS_A      ; Clear display regs to Shut off LED display
01A0 0068      00602          CLRF    DISPSEGS_B      ; Clear display regs to Shut off LED display
01A1 0069      00603          CLRF    DISPSEGS_C      ; Clear display regs to Shut off LED display
01A2 006A      00604          CLRF    DISPSEGS_D      ; Clear display regs to Shut off LED display
01A3 0B1E      00605          GOTO    main_loop       ; We are done - go back and do it again!
01A4           00606 new_display
01A4 00D9      00607          DECF    FLAGS,W         ; Move current mode state into W
01A5 0900      00608          CALL    mode_timer      ; Look-up register address of value to display
01A6 091F      00609          CALL    disp_value      ; Update display registers with new values
01A7 0B1E      00610          GOTO    main_loop       ; We are done - go back and do it again!
               00611
               00612 ; ************************************
               00613 ; * Set up and initialize the processor  *
               00614 ; ************************************
01A8           00615 init
01A8 0C03      00616          MOVLW   OPTION_SETUP    ; Place option reg setup into W
01A9 0002      00617          OPTION                  ; Set up OPTION register
01AA 0C05      00618          MOVLW   PORTA           ; Place beginning of RAM/Port location into W
01AB 0024      00619          MOVWF   FSR             ; Now initialize FSR with this location
01AC           00620 clear_mem
01AC 0060      00621          CLRF    INDADDR         ; Clear the FSR pointed memory location
01AD 03E4      00622          INCFSZ  FSR,F           ; Point to the next location
01AE 0BAC      00623          GOTO    clear_mem       ; Jump back to clear memory routine
01AF 0572      00624          BSF     ALM_HOUR_LD,3   ; Place 8:00 into alarm register
01B0 02AE      00625          INCF    CLK_HOUR_LD,F   ; Place 1:00 into clock register
01B1 0CEE      00626          MOVLW   0EEh            ; Turn on display A scan line, others off
01B2 003B      00627          MOVWF   PREVSCAN        ;
01B3 0040      00628          CLRW
01B4 0006      00629          TRIS    PORTB           ; Make all Port B pins outputs.
01B5 0005      00630          TRIS    PORTA           ; Make all Port A pins outputs.
01B6 0539      00631          BSF     FLAGS,1         ; Set up current mode to CLOCK, display ON
01B7 0519      00632          BSF     FLAGS,0
01B8 04A3      00633          BCF     ALARMOK         ; Don't want to trigger alarms
01B9 04C3      00634          BCF     EGGOK
01BA 059D      00635          BSF     DISPON          ; Turn on the displays
01BB           00636 mfg_checkkey
01BB 0966      00637          CALL    scan_keys       ; Lets see what is pressed
01BC 07D8      00638          BTFSS   UPKEY           ; Goto self-test if UP key is pressed at pwr up
01BD 0B1E      00639          GOTO    main_loop       ; Normal operation - Jump to the main loop
               00640
               00641 ; ************************************************************
               00642 ; * Self-test code for manufacturing only - test buttons and LEDs *
               00643 ; ************************************************************
01BE           00644 mfg_selftest
01BE 0C70      00645          MOVLW   b'01110000'     ; Place all key on pattern into W
01BF 002D      00646          MOVWF   CLK_MIN_HD      ; Use CLK_MIN_HD for keystuck ON test
```


© 1997 Microchip Technology Inc.

```
01C0 006F       00647          CLRF    CLK_HOUR_HD     ; Use CLK_HOUR_HD for keystuck OFF test
01C1            00648 mfg_display
01C1 020B       00649          MOVF    CLK_SEC,W       ; Current segment display count -> W
01C2 0949       00650          CALL    mfg_led_lookup  ; Look-up the next segment pattern to display
01C3 0026       00651          MOVWF   PORTB           ; Move the pattern to PORT B to display it
01C4            00652 mfg_timer
01C4 0201       00653          MOVF    TMR0,W          ; Place current TMR0 value into W
01C5 019A       00654          XORWF   PREVTMR0,W      ; Lets see which bits have changed...
01C6 003C       00655          MOVWF   TEMP            ; All changed bits are placed in temp for test
01C7 01BA       00656          XORWF   PREVTMR0,F      ; Update Previous TMR0 value.
01C8 07FC       00657          BTFSS   TEMP,7          ; Skip if it is not time to increment second
01C9 0BC4       00658          GOTO    mfg_timer       ; It is not time to move to next digit - go back
01CA 02AB       00659          INCF    CLK_SEC,F       ; Move to the next display pattern
01CB            00660 mfg_check_digit
01CB 07AB       00661          BTFSS   CLK_SEC,5       ; Skip if we have timed out waiting for button
01CC 0BD5       00662          GOTO    mfg_doneclk     ; Jump to check for the next button press
01CD            00663 mfg_nextdigit
01CD 006B       00664          CLRF    CLK_SEC         ; Clear out timer
01CE 0906       00665          CALL    buzz_now        ; Send out a buzzer beep!
01CF 077B       00666          BTFSS   PREVSCAN,3      ; Skip if we have NOT tested the last digit
01D0 0BE5       00667          GOTO    finish_mfg_test ; Jump to the end after last digit tested
01D1 035B       00668          RLF     PREVSCAN,W      ; Select the next digit through a rotate..
01D2 037B       00669          RLF     PREVSCAN,F
01D3 021B       00670          MOVF    PREVSCAN,W      ; Place next digit select into W
01D4 0025       00671          MOVWF   PORTA           ; Update port A to select next digit
01D5            00672 mfg_doneclk
01D5 0966       00673          CALL    scan_keys       ; Scan the keys to see what is pressed...
01D6 0218       00674          MOVF    KEYPAT,W        ; Place pattern into W
01D7 016D       00675          ANDWF   CLK_MIN_HD,F    ; Make sure keys are not stuck ON
01D8 012F       00676          IORWF   CLK_HOUR_HD,F   ; Make sure each key is pressed at least once
01D9 077B       00677          BTFSS   PREVSCAN,3      ; Skip if we are NOT at the last digit
01DA 05F8       00678          BSF     KEYPAT,7        ; Set flag bit to indicate we are done!
01DB 0C08       00679          MOVLW   .8              ; Place 8 into W
01DC 008B       00680          SUBWF   CLK_SEC,W       ; CLK_SEC - W => W
01DD 0703       00681          BTFSS   STATUS,C
01DE 0078       00682          CLRF    KEYPAT
01DF 03B8       00683          SWAPF   KEYPAT,F
01E0 025B       00684          COMF    PREVSCAN,W
01E1 0158       00685          ANDWF   KEYPAT,W
01E2 0743       00686          BTFSS   STATUS,Z
01E3 0BCD       00687          GOTO    mfg_nextdigit
01E4 0BC1       00688          GOTO    mfg_display
01E5            00689 finish_mfg_test
01E5 022D       00690          MOVF    CLK_MIN_HD,F
01E6 0743       00691          BTFSS   STATUS,Z
01E7 0BEF       00692          GOTO    bad_switch
01E8 020F       00693          MOVF    CLK_HOUR_HD,W
01E9 0F70       00694          XORLW   070h
01EA 0743       00695          BTFSS   STATUS,Z
01EB 0BEF       00696          GOTO    bad_switch
01EC            00697 mfg_cleanup
01EC 006F       00698          CLRF    CLK_HOUR_HD     ; Restore temp registers to zero
01ED 006D       00699          CLRF    CLK_MIN_HD      ; Restore temp registers to zero
01EE 0B1E       00700          GOTO    main_loop       ; Jump to main loop
01EF            00701 bad_switch
01EF 026D       00702          COMF    CLK_MIN_HD,F
01F0 038D       00703          SWAPF   CLK_MIN_HD,W
01F1 0038       00704          MOVWF   KEYPAT
01F2 05EF       00705          BSF     CLK_HOUR_HD,7
01F3 038F       00706          SWAPF   CLK_HOUR_HD,W
01F4 0178       00707          ANDWF   KEYPAT,F
01F5 0C7F       00708          MOVLW   07Fh
01F6 0026       00709          MOVWF   PORTB
01F7 006C       00710          CLRF    CLK_MIN_LD
01F8 05AC       00711          BSF     CLK_MIN_LD,5
01F9            00712 loop_bad_sw
```

5

© 1997 Microchip Technology Inc.

```
01F9 0907       00713          CALL    buzz_now_dispon ; Beep the buzzer constantly for a few secs
01FA 02EC       00714          DECFSZ  CLK_MIN_LD,F    ; Decrement counter and skip when done
01FB 0BF9       00715          GOTO    loop_bad_sw     ; Not done buzzing - go back and do it again
01FC 0BEC       00716          GOTO    mfg_cleanup     ; Done buzzing - clean-up and run clock
                00717          END


MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0100 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0140 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0180 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXX--X
0F80 : ---------------- ---------------- ---------------- ----------------
0FC0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.


Errors   :     0
Warnings :     0
Messages :     0
```


© 1997 Microchip Technology Inc.

SECTION 6
SECURE DATA PRODUCT APPLICATION NOTES

Converting NTQ104/105/106 Designs to HCS200/300s - AN644 6-1
PIC16C57 Based Code Hopping Security System - AN645 6-7
KEELOQ® Code Hopping Decoder Using a PIC16C56 - AN661 6-19
KEELOQ® Code Hopping Decoder Using Secure Learn - AN662 6-33
KEELOQ® Simple Code Hopping Decoder - AN663 6-49
Using KEELOQ® to Generate Hopping Passwords - AN665 6-63

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

AN644

Converting NTQ104/105/106 Designs to HCS200/300s

Author: Kobus Marneweck
Microchip Technology Inc.

INTRODUCTION

NTQ104/105/106 encoders have been used in many designs requiring the high security offered by the KEELOQ® code hopping system. With the introduction of Microchip's encoders, the HCS200/300 and the high-voltage HCS301, designers can do seamless conversions of their NTQ104/105/106 encoders to the new Microchip HCS200/300 devices. In addition, designers can convert their NTQ104/105/106 encoders to the new Microchip devices and take advantage of the extra features and lower cost of the new devices with minimal changes. This application note will help designers to do these conversion and take advantage of the features on the HCSXXX series to reduce the required component count.

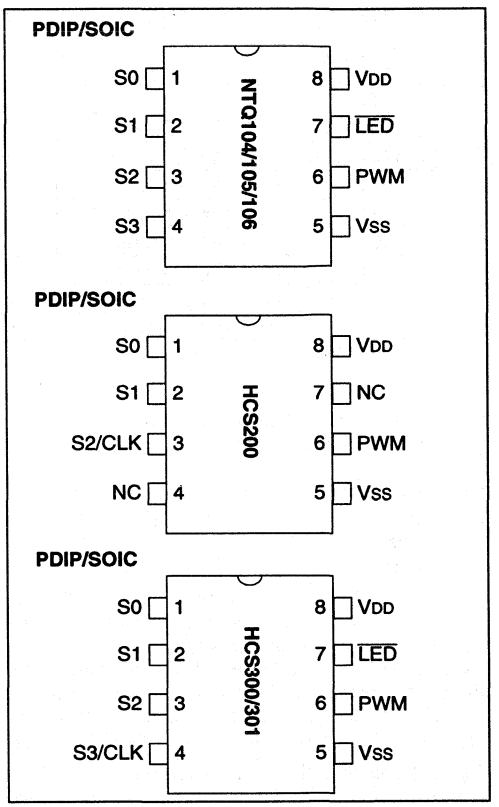


TABLE 1: SUMMARY OF FEATURES

Encoder	Function Codes	Operating Voltage	Programming Lines	Transmission Length
HCS300	15	2.0 to 6.3V	Clock + Data	66
HCS301	15	4.5 to 12.5V	Clock + Data	66

Note: The NTQ devices used a single bi-directional PWM line for programming, but required S3 to be pulled high to select programming mode. The NTQ and HCS devices, therefore, both require a 4-pin connection to program, but no external diodes are used.

KEELOQ is a registered trademark of Microchip Technology Inc.

© 1997 Microchip Technology Inc.

TABLE 2: FUNCTIONS

Buttons Pressed S3 S2 S1 S0	NTQ104/106 Function Code	NTQ105 Function Code	HCS200 Function Code	HCS300/301 Function Code
0000	Reset	Reset	Reset	Reset
0001	X001	X001	X001	0001
0010	X010	X010	X010	0010
0011	X011	X011	X011	0011
0100	X100	Not supported	X100	0100
0101	X101	Not supported	X101	0101
0110	X110	Not supported	X110	0110
0111	X111	Not supported	X111 (Note)	0111
1000	Not supported	Not supported	Not supported	1000
1001	Not supported	Not supported	Not supported	1001
1010	Not supported	Not supported	Not supported	1010
1011	Not supported	Not supported	Not supported	1011
1100	Not supported	X100	Not supported	1100
1101	Not supported	X101	Not supported	1101
1110	Not supported	Not supported	Not supported	1110
1111	Not supported	Not supported	Not supported	1111 (Note)

Note: Seed transmission mode.

CONVERTING LOW VOLTAGE (3.0V TO 6.0V) DESIGNS

Low voltage designs do not require changes to the board layout since the NTQ and HCS devices are pin compatible. The HCS300 will support all the function codes available on the NTQ104/105/106.

The HCS200 can be used to replace the NTQ104/106, except where function X111 was used. Most designs, however, do not use function X111 and, therefore, can benefit from the extra functionality — in the HCS200, this button combination is used for the Secure Learning mode. The circuit diagram (Figure 1) shows the replacement of the NTQ devices with the HCS devices.

Figure 1 shows a NTQ transmitter design where two diodes were used to allow programming with a 3-pin probe. This is not an optimum implementation since the diodes are only used during programming. The HCS300 circuit shows how the design can be simplified. The HCS200 does not have an $\overline{\text{LED}}$ output but $\overline{\text{LED}}$ power indication can be implemented by connecting the $\overline{\text{LED}}$ in line with the RF power or in-line with the HCS200 power. The clock pin for the HCS200 is on pin three. Interchangeability between the HCS200 and HCS300 can be achieved by using pin three for the clock in the HCS300 circuit.

Figure 2 shows a typical multi-button transmitter design using the NTQ104/105/106. In this case no changes are required for the HCS300. The HCS200 will require the $\overline{\text{LED}}$ connected in-line of power indication and the clock line connected to pin three.

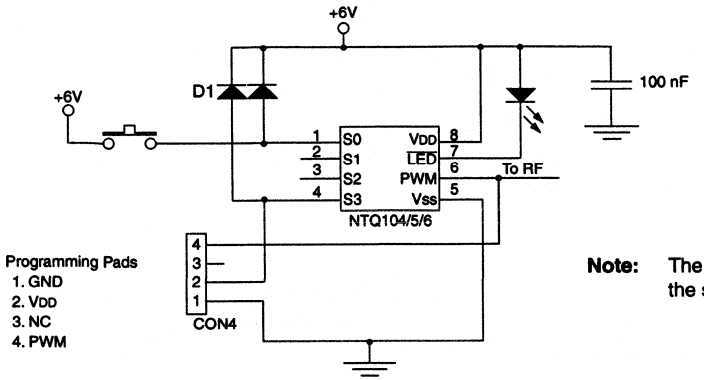
CONVERTING HIGH VOLTAGE (6.0V TO 12.0V) DESIGNS

The HCS301 has an on-board regulator that enables the device to operate with a 12V supply. This allows a significant reduction in external components. The steering diodes, protection resistors and zener diode are no longer required. Resistor R1 can be used to reduce the operating current. The PWM output voltage will swing between ground and V_{DD}. The circuit diagram (Figure 3) shows the simplified circuit.

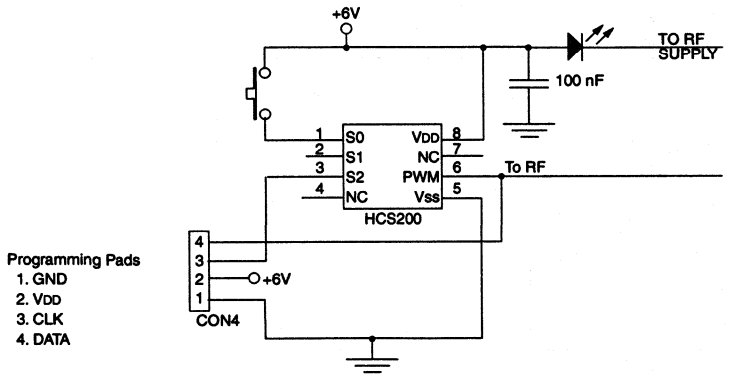
The HCS300/301 can also be used in the same circuit without changes.

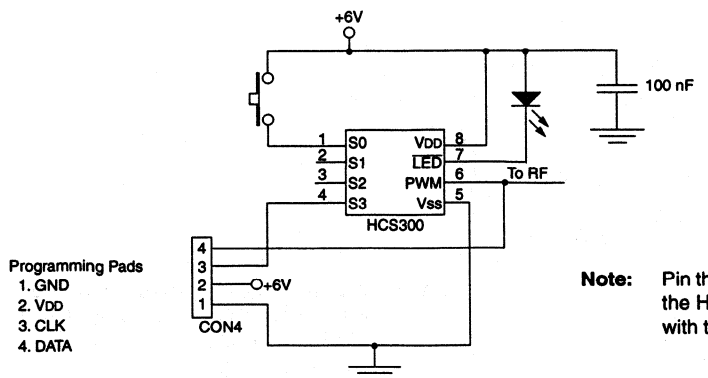
© 1997 Microchip Technology Inc.

FIGURE 1: CONVERTING NTQ104/105/106 TO HCS200/300



Note: The HCS300/301 can also be used in the same circuit with D1 removed.

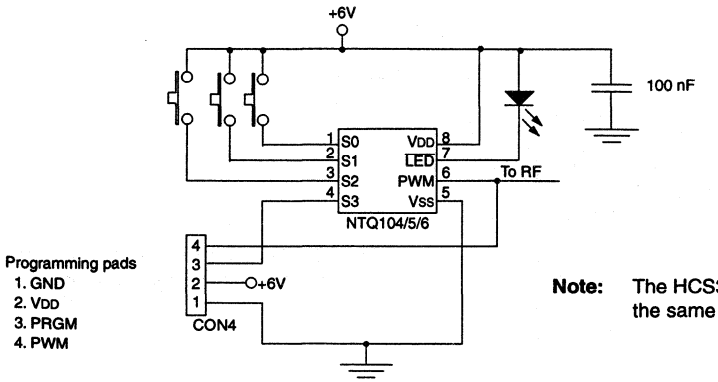




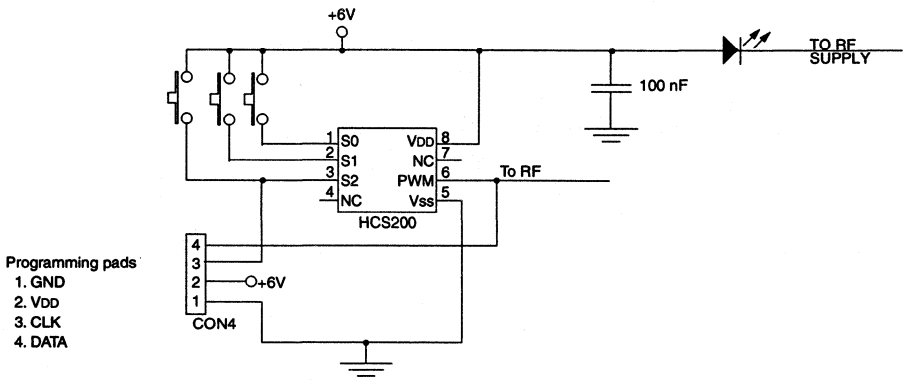
Note: Pin three can also be used as a clock on the HCS300 to provide pin compatibility with the HCS200.

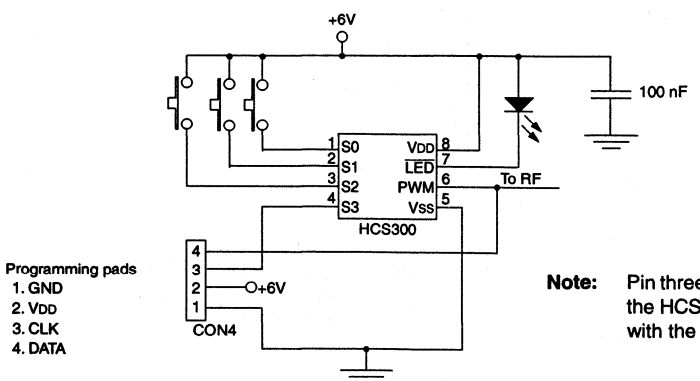
6

FIGURE 2: CONVERTING NTQ104/105/106 TO HCS200/300



Note: The HCS300/301 can also be used in the same circuit.

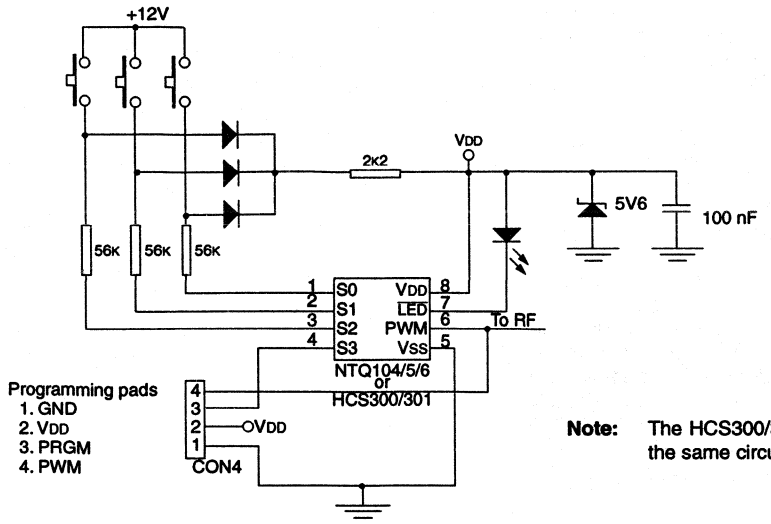




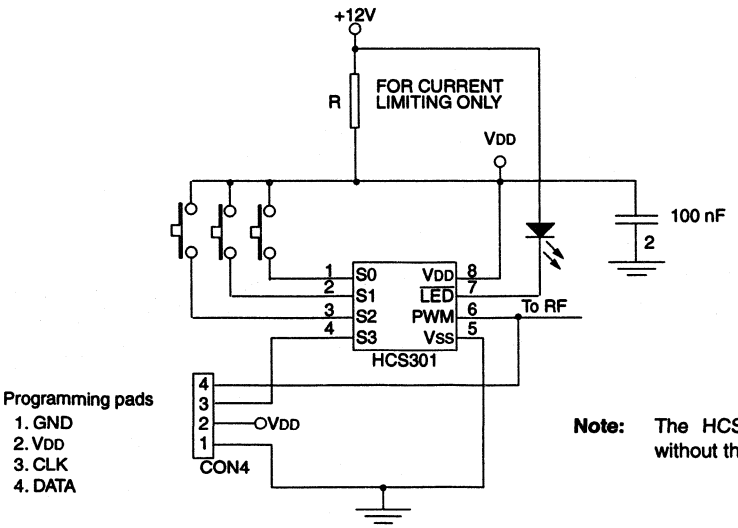
Note: Pin three can also be used as a clock on the HCS300 to provide pin compatibility with the HCS200.

© 1997 Microchip Technology Inc.

FIGURE 3: CONVERTING NTQ104/105/106 TO HCS301



Note: The HCS300/301 can also be used in the same circuit.



R = 1.5k for 12V battery.
R = 0R for 9V battery.

Note: The HCS301 must be programmed without the battery connected.

6

PROGRAMMING

It is suggested that programming be done without the battery connected. The programming probe supplies 5V during programming and damage to the probe may occur if for instance the transmitter is powered with a 12V battery.

COMPATIBILITY WITH DECODERS

NTQ109/NTQ110 Decoders

These decoders only receive 56-bit codes. Transmissions from HCS200/300 transmitters are, therefore, truncated to 56 bits. With the NTQ109/110 option selected in the programming software supplied with the PG306001 programmer, HCS200/300 transmitters will be programmed with a 24-bit serial number meaning the higher than 24 bits will all be zero. The 24-bit serial number will be used for key generation. An HCS200/300 transmitter is fully compatible with NTQ109/110 decoders if programmed correctly.

PIC16C56 and EEPROM Based Decoders, and HCS509 Decoder

PIC16C56 based HCS509 decoders automatically distinguish between NTQ104/105/106 56-bit transmissions and HCS200/300 66-bit transmissions. The decoder will generate the key from the longer 28-bit serial number if a 66-bit transmission is detected.

© 1997 Microchip Technology Inc.

MICROCHIP

AN645

PIC16C57 Based Code Hopping Security System

Author: *Kobus Marneweck*
Microchip Technology Inc.

OVERVIEW

This document describes a PIC16C57 based code hopping automotive security system. The security system implements all the basic features found on security systems and can be changed to modify or add features as required. The code can also be moved to a higher functionality PICmicro™ microcontroller for more I/O or code space.

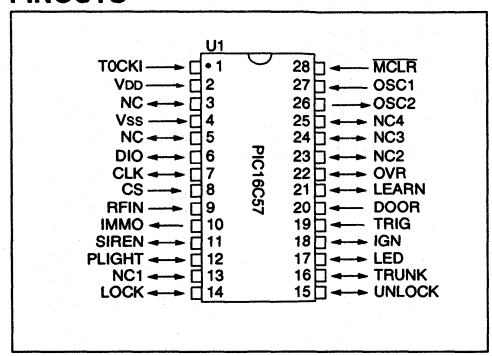
FEATURES

- Code hopping alarm system
- System can handle up to six transmitters
- Learning of new transmitters
- Arm/Disarm
- Trunk release
- Car finder
- Panic
- Locking/unlocking of doors
- Door and shock sensor trigger inputs

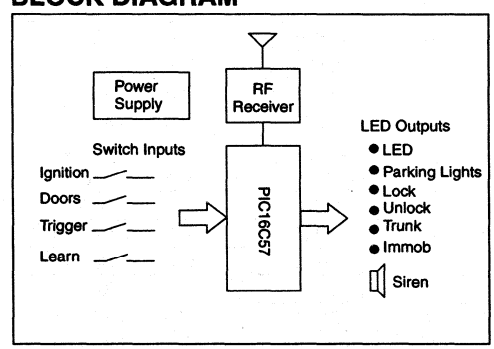
RECOMMENDED READING

If the reader is unfamiliar with KEELOQ Code Hopping it would be helpful to read *Introduction to KEELOQ* (DS91002). This and other KEELOQ literature can be found on Microchip's BBS, Web site or from a Microchip field application engineer. The software described in this application note is available on a diskette from Microchip by ordering DS40149. A complete list of KEELOQ literature can be found at the end of the application note.

PINOUTS



BLOCK DIAGRAM



6

© 1997 Microchip Technology Inc.

MEMORY MAP EEPROM (16 BIT WORDS)

Address		Address	
00h	USER0	20h	CNT20
01h	LRN_PTR	21h	CNT21
02h	BSTATUS	22h	SER20
03h	SSTATUS	23h	SER21
04h	TMPCNT	24h	KEY20
05h	USER1	25h	KEY21
06h	USER2	26h	KEY22
07h	USER3	27h	KEY23
08h	USER4	28h	CNT30
09h	USER5	29h	CNT31
0Ah	DIS0	2Ah	SER30
0Bh	DIS1	2Bh	SER31
0Ch	DIS2	2Ch	KEY30
0Dh	DIS3	2Dh	KEY31
0Eh	DIS4	2Eh	KEY32
0Fh	DIS5	2Fh	KEY33
10h	CNT00	30h	CNT40
11h	CNT01	33h	CNT41
12h	SER00	32h	SER40
13h	SER01	33h	SER41
14h	KEY00	34h	KEY40
15h	KEY01	35h	KEY41
16h	KEY02	36h	KEY42
17h	KEY03	37h	KEY43
18h	CNT10	38h	CNT50
19h	CNT11	39h	CNT51
1Ah	SER10	3Ah	SER50
1Bh	SER11	3Bh	SER51
1Ch	KEY10	3Ch	KEY50
1Dh	KEY11	3Dh	KEY51
1Eh	KEY12	3Eh	KEY52
1Fh	KEY13	3Fh	KEY53

LRN_PTR – Learn pointer points to the next available learn position.

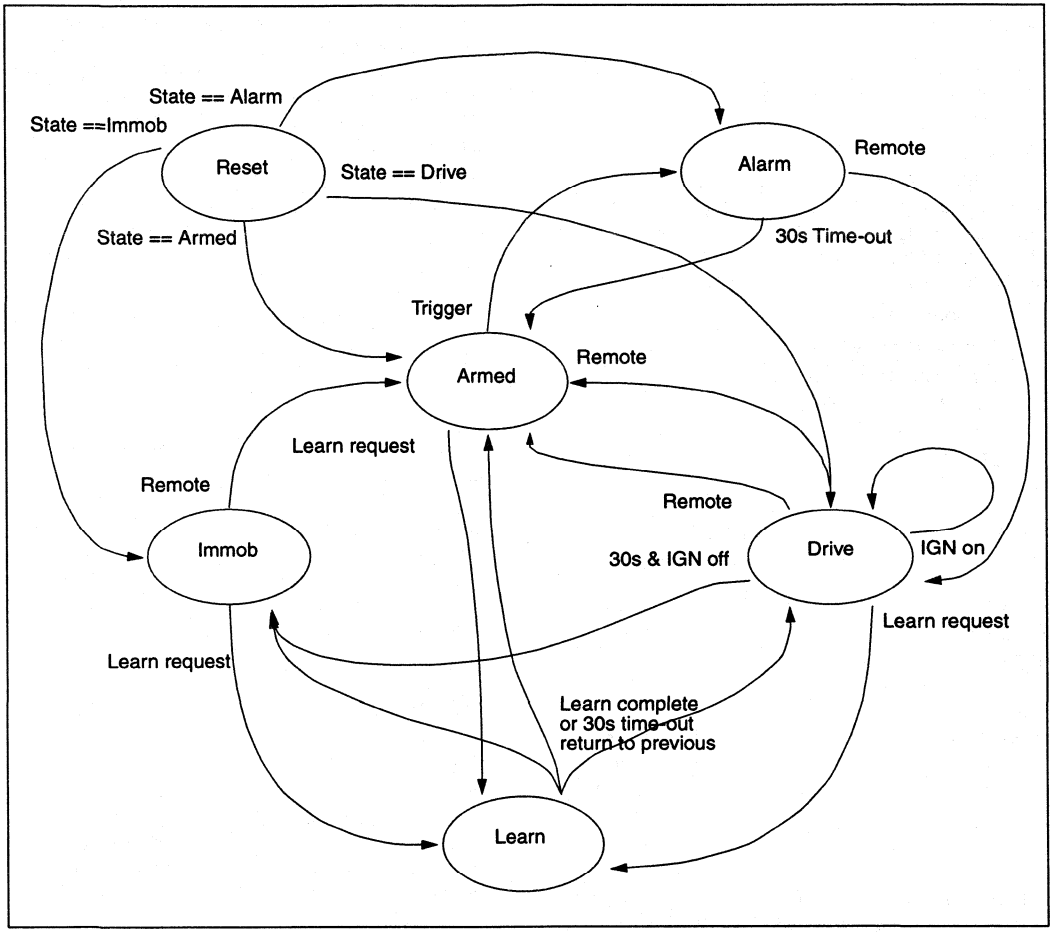
SSTATUS – Stores the system status.

BSTATUS – Backup copy of system status.

TMPCNT – Stores the temporary counter for resynchronization.

© 1997 Microchip Technology Inc.

FIGURE 1: ALARM STATE DIAGRAM



© 1997 Microchip Technology Inc.

OPERATION

Reset

Reset initializes the I/O ports, variables, and flags. The system status is read from EEPROM and the status is restored.

Armed

When the system enters armed state, the doors are locked (activate LOCK) and the SIREN and PLIGHT are activated for 50 ms. The LED changes to a slow flash rate. If a trigger is detected (IGN, DOOR or TRIGGER) the system changes to the alarm state.

Actions upon entry:

1. Flash parking lights for 50 ms.
2. Chirp siren for 50 ms.
3. Lock doors for 500 ms.
4. Update system status.
5. LED flash.
6. Disable start.

TABLE 1: STATE CHANGE TABLE

Condition	Next State
IGN high	Alarm
TRIG high	Alarm
DOOR high	Alarm
Panic (any button activated for 2 seconds)	Alarm
Remote function 1	Drive
Remote function 2 (trunk release)	Armed
Remote function 3 (car finder)	Armed
LEARN high	Learn

Alarm

Alarm state is entered whenever a trigger is detected in armed state. SIREN is activated and PLIGHT is turned on and off at a 1 Hz rate. If a remote is detected in this state, the system changes to drive state. After a 30-second delay, SIREN and PLIGHT will be deactivated and the system returned to armed state.

Actions upon entry:

1. Flash parking lights.
2. Siren on.
3. LED flash.
4. Update system status.
5. Disable start.

TABLE 2: STATE CHANGE TABLE

Condition	Next state
Panic (any button activated for 2 seconds)	Alarm
Remote function 1	Drive
Remote function 2 (trunk release)	Armed
30-second timeout	Drive

Drive

When the system enters drive state, the doors are unlocked (activate UNLOCK), and the SIREN and PLIGHT are activated twice for 50 ms.The IMMOB output is activated to enable the starting of the vehicle and LED is turned off. A remote signal will return the system to armed state.

Actions upon entry:

1. Flash parking lights for 50 ms.
2. Chirp siren for 50 ms.
3. Unlock doors for 500 ms.
4. Flash parking lights for 50 ms.
5. Chirp siren for 50 ms.
6. Update system status.
7. LED off.
8. Enable start.

TABLE 3: STATE CHANGE TABLE

Condition	Next State
Panic (any button activated for 2 seconds)	Alarm
Remote function 1 & IGN low	Armed
Remote function 1 & IGN high	Drive
Remote function 2 (trunk release)	Drive
Remote function 3 (car finder)	Drive
30-second timeout & IGN off	Immob
LEARN high	Learn

Immob

If the IGN is turned off for more than 30 seconds, the system will immobilize. The IMMOB output is turned off, and the LED is turned on. A remote signal only will change the state to armed, and a remote signal with the IGN on will return to drive state.

Actions upon entry:

1. Update system status.
2. LED off.
3. Disable start.

TABLE 4: STATE CHANGE TABLE

Condition	Next State
Panic (any button activated for 2 seconds)	Alarm
Remote function 1 & IGN low	Armed
Remote function 1 & IGN high	Drive
Remote function 2 (trunk release)	Immob
Remote function 3 (car finder)	Immob
LEARN high	Learn

© 1997 Microchip Technology Inc.

Learn

A LEARN input in any state will put the system in learn mode. After learn is completed or timed out the system returns to the previous state.

Actions upon entry:

1. Update system status—set PASS1.
2. LED on.

After first transmission:

1. Update system status—et PASS2.
2. LED off.

After second transmission:

1. Update system status—set NORMAL.
2. LED on for 1 second.
3. Return to previous state.

TABLE 5: STATE CHANGE TABLE

Condition	Next State
Remote first operation	Pass2
Remote second operation	Return to previous state
LEARN high for 8 seconds	Erase all transmitters

FUNCTIONAL MODULES

Reception

The reception routine is based on reliable algorithms used in previous implementations of KEELOQ decoders. Automatic baud rate detection is used to compensate for variations in baud rate from different encoders of a specific type as well as the difference in baud rate between different encoders (HCS200, HCS300, HCS301, HCS360, HCS361, and NTQ106). The reception routine will be able to handle 56- and 66-bit transmissions. The reception routine will determine the type of transmission by the number of bits in the transmission. This routine will be the same for all implementations.

Key Generation and Decryption

Decryption is done in software in the implementation. The decryption and key generation algorithms is implemented in software. The manufacturer's key is stored in program memory and code protected to securely store the key.

Validation

Validation consists of the following steps:

1. Checking the serial number (24 or 28 bits) against the stored transmitters.
2. Comparing the discrimination value (12 bits) against the stored discrimination value.
3. Checking that the synchronization counter falls within the first synchronization window.
4. Checking if the synchronization counter falls within the second synchronization window.
5. If found to be correct, updating the synchronization counter.

TABLE 6: Function Interpretation

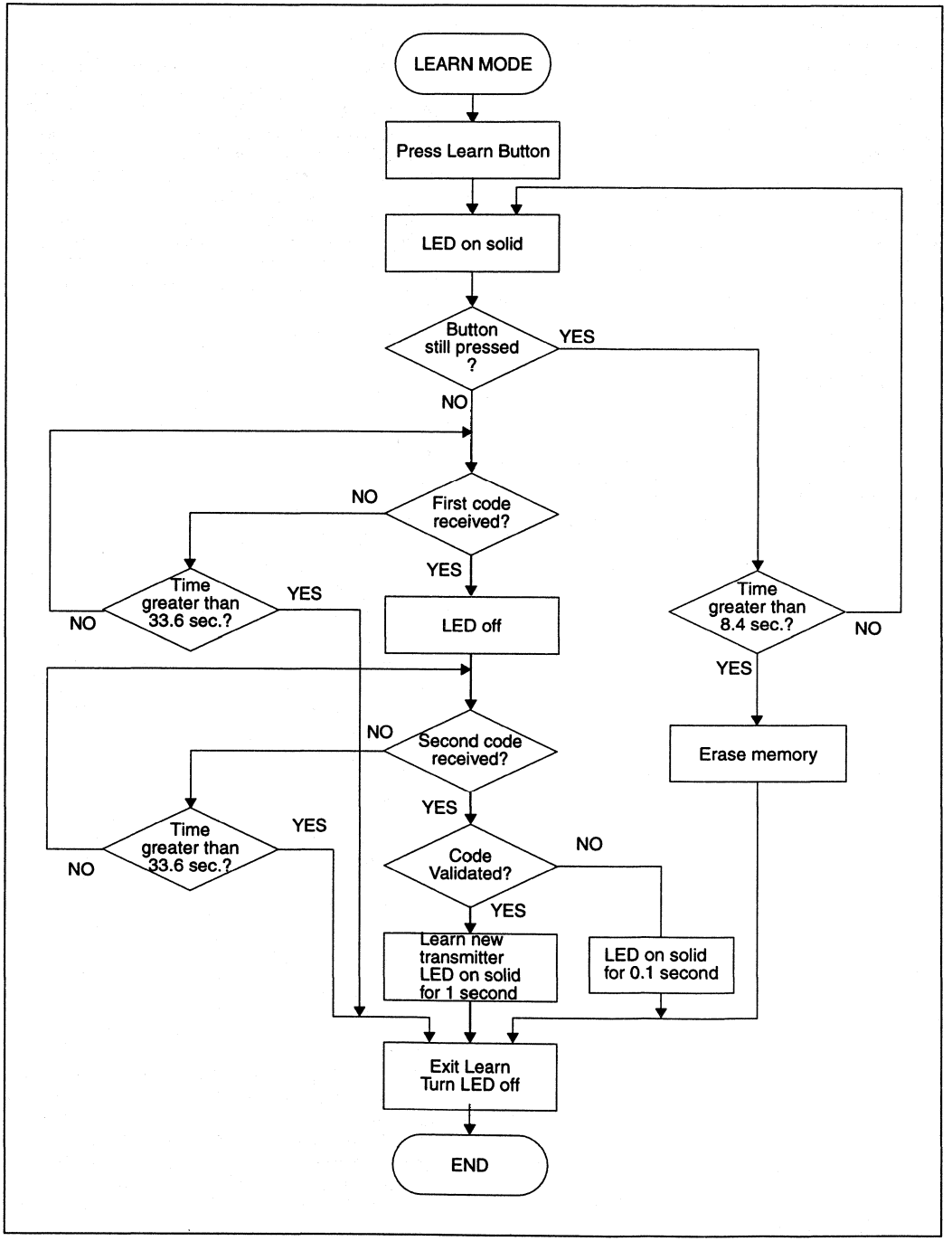
Transmitter Button	Function Code	System Function
1	0001	Arm/Disarm
2	0010	Trunk release
3	0011	Car finder
1, 2 or 3 for 2 seconds	00XX	Panic

Learn

The LEARN input is active high. Learning is initiated by momentarily pressing the LEARN button. The decoder uses the current learning position as a scratch pad area. This means that an unsuccessful learn will delete the information stored at that learn position. The learn pointer will not be incremented if the learn was unsuccessful. The flow chart (Figure 2) shows the learning operation.

© 1997 Microchip Technology Inc.

FIGURE 2: LEARN OPERATION



© 1997 Microchip Technology Inc.

The following checks will be performed on the received codes to determine if the transmitter is valid:

1. The first code that is received is checked for bit integrity.
2. The stored serial numbers are searched to check if a transmitter is relearned. If a relearn is taking place, that position is used. Otherwise, the position pointed to by the learn pointer will be used.
3. The serial number is stored in the current learn position and used to generate a key.
4. The hop code is decrypted and the result stored temporarily.
5. The serial number of the second code that is received will be compared to the first received serial number.
6. The second hop code is decrypted and the discrimination values compared.
7. The synchronization counters of the decrypted codes will be compared to check that they are sequential codes.
8. If all the checks pass the learn were successful, the learn pointer is incremented. Otherwise, the position is erased.

Operation

1. Press and release the LEARN button. Indicator LED will turn on to indicate learn mode.
2. Press transmitter button. The LED will turn off.
3. Press transmitter a second time. The LED will turn on for 1 second to indicate that the transmitter was learned successfully.
4. Repeat steps 1-3 to learn up to six transmitters. The seventh transmitter will overwrite the first transmitter that was learned.
5. Learn will be terminated if two nonsequential codes were received or if two acceptable codes were not decoded within 33.6 seconds. A valid learn will be indicated by the LED turning on solid for 1 second.
6. Erasing all the transmitters is accomplished by pressing and holding the LEARN button for 8.4 seconds. The LED will turn off at the end of the 8.4 seconds to indicate that the transmitters were erased. The learn pointer will be reset to the first position.

© 1997 Microchip Technology Inc.

TABLE 7: DEVICE PINOUT

PIN	PIC16C57 Function	Alarm Function	PIN	PIC16C57 Function	Alarm Function
1	RTCC	APP select	28	$\overline{\text{MCLR}}$	RESET
2	VDD	+5V supply	27	Osc In	RC osc (4 MHz)
3	NC		26	Osc Out	
4	GND	Ground	25	Port C Bit 7	NC
5	NC		24	Port C Bit 6	NC
6	Port A Bit 0	EEPROM DIO(3+4)	23	Port C Bit 5	NC
7	Port A Bit 1	EEPROM CLK (2)	22	Port C Bit 4	OVR
8	Port A Bit 2	EEPROM CS (1)	21	Port C Bit 3	LEARN
9	Port A Bit 3	RFIN	20	Port C Bit 2	DOOR
10	Port B Bit 0	IMMOB	19	Port C Bit 1	TRIG
11	Port B Bit 1	SIREN	18	Port C Bit 0	IGN
12	Port B Bit 2	PLIGHT	17	Port B Bit 7	LED
13	Port B Bit 3	NC	16	Port B Bit 6	TRUNK
14	Port B Bit 4	LOCK	15	Port B Bit 5	UNLOCK

TABLE 8: TIMING PARAMETERS

Parameter	Typical	Unit
Armed LED flash rate	1	per second
Siren time-out	33	second
Drive time-out	33	second
Learn time-out	33	second
All erase	8	second
LOCK, UNLOCK, TRUNK activation	500	ms
Siren chirp (arm & disarm)	50	ms
Parking light (arm & disarm)	50	ms
Parking light flash rate (siren)	1	per second
Panic	2	seconds

© 1997 Microchip Technology Inc.

KEELOQ LITERATURE

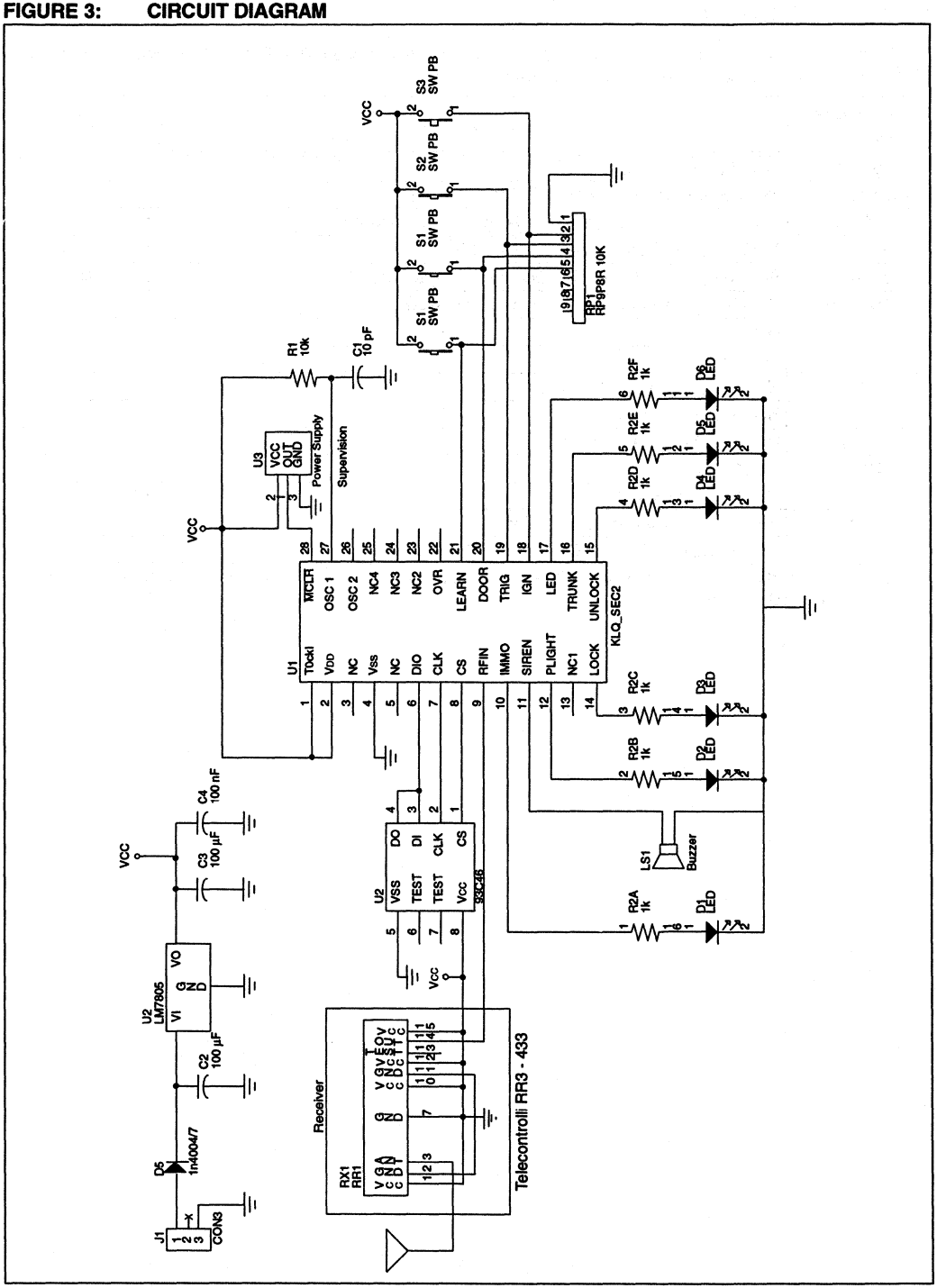
The following literature is available from Microchip's field application engineers, directly from Microchip, and on both the Microchip BBS and Web site unless otherwise specified.

Document Number	Description	Notes
DS91002	Introduction to KEELOQ	
DS1044	HCS300 Evaluation Kit User's Guide	
DS91000	Secure Learning RKE Systems Using KEELOQ Encoders	
DS00642	Code Hopping Decoder Using a PIC16C56	Note
DS00652	Secure Learn Code Hopping Decoder Using a PIC16C56	Note
DS00645	PIC16C57 Based Code Hopping Security System	Note
DS40138	HCS200 Data Sheet	
DS21137	HCS300 Data Sheet	
DS21143	HCS301 Data Sheet	
DS40152	HCS360 Data Sheet	
DS40146	HCS361 Data Sheet	
DS40147	HCS509 Data Sheet	
DS40151	HCS512 Data Sheet	
DS50136	Programming Station User's Guide	
DS00644	Converting NTQ105/106 Designs to HCS200/300s	

Note: The complete document includes software and is available on diskette. The diskette can be ordered by ordering DS40149.

© 1997 Microchip Technology Inc.

FIGURE 3: CIRCUIT DIAGRAM



© 1997 Microchip Technology Inc.

LIST OF CHANGES

Date	Version	Page	Paragraph	Change
08/16/96	1.0			Original

© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN661

Code Hopping Decoder Using a PIC16C56

Author: *Steven Dawson*
Microchip Technology Inc.

OVERVIEW

This application note fully describes the working of a code hopping decoder implemented on a Microchip PIC16C56 microcontroller. Background is given on the various KEELOQ® code hopping encoders that can be used with the decoder, the decoder hardware described, and descriptions of the various software modules comprising the system. The software can be used to implement a stand-alone decoder or integrated with full function security systems. The decoder supports the Microchip HCS200, HCS201, HCS300, HCS301, and NTQ106 KEELOQ Code Hopping Encoders.

KEY FEATURES

- Stand-alone decoder
- Compatible with Microchip HCS200, HCS201, HCS300, HCS301, and NTQ106 encoders
- Automatic baud rate detection
- Automatic encoder type detection
- Four function outputs
- Six learnable transmitters
- RC Oscillator

6

Notice:

This is a non-restricted version of Application Note AN642 which is available under the KEELOQ License Agreement. The license agreement can be ordered from the Microchip Literature Center as DS40149.

KEELOQ is a registered trademark of Microchip Technology Inc.

© 1997 Microchip Technology Inc.

INTRODUCTION TO KEELOQ ENCODERS

All KEELOQ encoders use the KEELOQ code hopping technology to make each transmission by an encoder unique. The encoder transmissions have two parts. The first part changes each time the encoder is activated and is called the hopping code part. The second part is the serial number of the encoder, identifying it to a decoder.

Hopping Code

The Hopping Code contains function information, a discrimination value, and a synchronization counter. This information is encrypted by an encryption algorithm before being transmitted. A 64-bit encryption key is used by the encryption algorithm. If one bit in the data that is encrypted changes, the result is that an average of half the bits in the output will change. As a result, the hopping code changes dramatically for each transmission and can not be predicted.

Function Information

The encoder transmits up to four bits of function information. Up to 15 different functions are available (0000 is related to the reset condition in all the current encoders, and can never be transmitted).

Discrimination Value

Stored in the encoder EEPROM, this information is used to check integrity of decryption operation in the decoder. If known information is inserted into the transmitted string before encryption, the same information can be used at the decoder to check whether the information has been decrypted correctly. In the NTQ106, 12 such bits are available, and 12 bits (including overflow bits) are available in the Microchip HCS encoders.

Synchronization Counters

The transmitted word contains a 16-bit synchronization counter. The synchronization information is used at the decoder to determine whether a transmission is valid, or a repetition of a previous transmission. Previous codes are rejected to safeguard against code grabbers. The HSC300/301 encoder transmits two overflow bits which may be used to extend the range of the synchronization counter from 65,536 to 196,608 button operations.

Fixed Code

Serial Number

The encoder's serial number is transmitted every time the button is pressed. The serial number is transmitted unencrypted as part of the transmission, and serves to identify the encoder to the decoder. The number can be used during learning operations to calculate the key to be used for decrypting the transmissions.

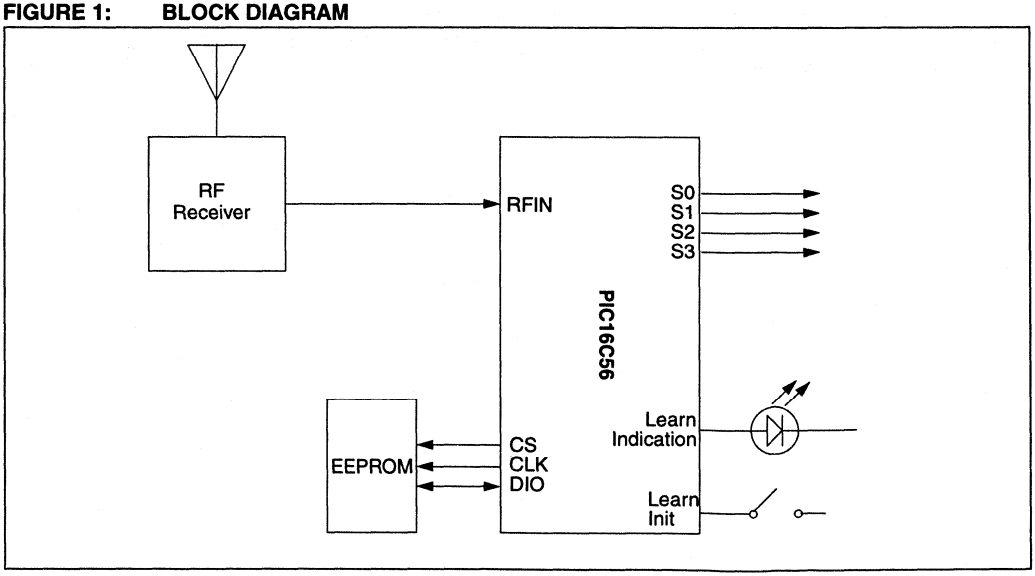
Other Status and Function Information

The HCS300/301 encoders include provision for four bits of function information and two status bits in the fixed code portion of its transmission. The two status bits indicate whether a repeated transmission is being sent, and whether the battery voltage is low. The HCS200/201 does not send repeated transmission information and the bit is permanently set to '0'.

Transmission Format

Table 1 contains a summary of the information contained in transmissions from each of the KEELOQ encoders that can be learned by the Microchip decoder.

FIGURE 1: BLOCK DIAGRAM



© 1997 Microchip Technology Inc.

TABLE 1: KEELOQ ENCODER TRANSMISSION SUMMARY

	NTQ106	HCS200/201*	HCS300/301
Total transmission length	56 bits	66 bits	66 bits
Hopping code portion (total length)	32 bits	32 bits	32 bits
Function bits	4 bits	3 bits (+1*)	4 bits
Discrimination bits	12 bits	12 bits	12 bits
Synchronization	16 bits	16 bits	16 bits
Fixed portion (total length)	24 bits	34 bits	34 bits
Serial number	24 bits	28 bits	28 bits
Function bits	—	3 bits (+1*)	4 bits
Status bits			
Low Voltage Indicator	—	1 bit	1 bit
Repeated Transmission Indicator	—	—	1 bit

*The HCS200/201 transmissions are padded to retain compatibility with the HCS300/301 encoders.

The code hopping portion is always transmitted first (LSB to MSB), followed by the fixed portion (LSB to MSB).

TABLE 2: HCS200/201* AND HCS300/301 TRANSMISSION FORMAT

Hop code	Serial number	Function	Status
LSB 32 bits MSB	LSB 28 bits MSB	4 bits	2 bits

*The HCS200/201 transmissions are padded to retain compatibility with the HCS300/301 encoders

TABLE 3: NTQ106 TRANSMISSION FORMAT

Hop code	Serial number
LSB 32 bits MSB	LSB 24 bits MSB

Bits are transmitted from left to right (i.e., the code hopping first).

PWM Format

In general, all KEELOQ encoders share a common transmission format.

- A preamble to improve biasing of decision thresholds in super-regenerative receivers. The preamble consists of alternate on and off periods, each lasting as long as a single elemental period.
- A calibration header, consisting of a low period of 10 elemental periods. Calibration actions should be performed on the low period of the header to ensure correct operation with header chopping.
- A string of 56 or 66 pulse width modulated bits, each consisting of three elements. The first element is high, the second contains the data transmitted and is either high or low, the third element is always low.
- A guard period is usually left between the transmissions. During this period nothing is transmitted by the encoder.

Figure 2 shows the sampling points when sampling data. The first and last elements are used exclusively to verify the integrity of the received signal. The first element (sample point A) is always high, the second (sample point B) is the complement of the data bit being sent, and the final element (sample point C) is always low. Because the period between the low portion of a bit (sample point C) and the rising edge of the following bit (sample point X) can vary somewhat, the rising edge of the first element (sample point X) is used to resynchronize the receiving routine to each incoming bit.

If random noise is being received, the probability of a set of three samples producing a valid combination is only 2^{-2} = 1/4. For a string of 56 bits, the corresponding figure is $2^{-112} > 2 \times 10^{-34}$. For longer strings, the probability is considerably less.

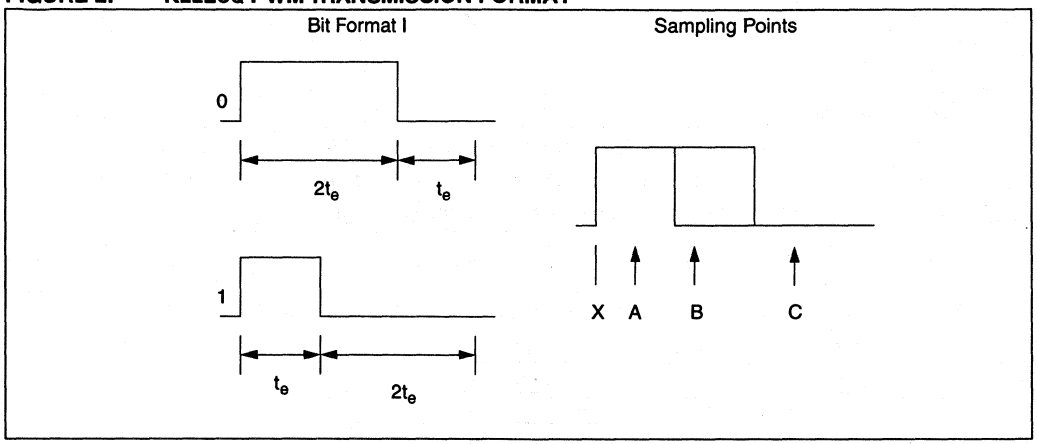
Integrity checking on incoming signals is important. Code hopping signals require significant processing, as well as EEPROM access, to decrypt. Unnecessary processing can be avoided by not attempting to decrypt incoming codes that have bit errors.

© 1997 Microchip Technology Inc.

TABLE 4: TRANSMISSION FORMATS

Encoder	Preamble	Header Chopping	Code Hopping Length	Fixed Code Length	Guard Period
NTQ106	—	—	32 bits	24 bits	3.4 ms
HCS200/201	•	•	32 bits	34 bits	6.4 ms
HCS300/301	•	•	32 bits	34 bits	6.4 ms

FIGURE 2: KEELOQ PWM TRANSMISSION FORMAT



IMPLEMENTATION

The Microchip decoder's primary hardware components are a PIC16C56 RISC microcontroller and a 93C46 EEPROM. However, this solution can be implemented in any PICmicro™ microcontroller with at least 1K words of programming. The operating frequency of the controller is 4 MHz. The microcontroller is used to capture transmissions from the various encoders, decrypt transmissions captured, and check the validity of the transmission based on the information in the decrypted transmission and information stored in the EEPROM. If a transmission from a valid encoder is received, the Microchip decoder activates the outputs dictated by the transmission.

Encoder information, such as serial number, synchronization information, and decryption key are stored externally in an EEPROM. The EEPROM used is a Microchip 93C46 CMOS serial EEPROM. The information stored in the EEPROM is encrypted to protect the decoders from cloning. The EEPROM encryption is less secure than the KEELOQ code hopping algorithm.

A more secure implementation of the decoder would be to modify the software in the application note and use a PIC16CXX with an internal EEPROM such as a PIC16C84. In this way communication between the PIC16C56 and EEPROM cannot be monitored.

As can be seen from the section on encoder transmissions there are differences in the transmission formats of the different encoders that are compatible with the system. The following section summarizes how the differences in transmitted data are dealt with by the decoder.

As the **serial number** information follows after the code hopping portion of the transmission, any number of serial number bits can be received and processed. For example, to maintain full compatibility between the Microchip NTQ106 and HCS encoders transmissions, a specific manufacturer might decide to disregard all but the first 24 serial number bits. In the Microchip decoder described the complete serial number is stored for both the NTQ106 encoders (24 bits) and the HCS range of encoders (28 bits).

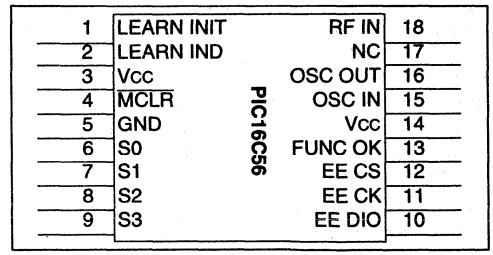
The serial number is used to identify the memory block used to store the 64-bit decryption key for a particular encoder because of the relationship between serial number and decryption key. In other words, the serial number is stored with the key. When a transmission is received, the decoder finds the correct memory block by checking all blocks until a matching serial number is found. The key is then retrieved from that particular memory block. A serial number of 0000000_{16} is considered invalid.

© 1997 Microchip Technology Inc.

Validation of a received transmission consists of two parts. The first includes checking the integrity of the decryption operation. Here the decoder compares the 12-bit discrimination value received with the stored discrimination value. This includes the full 12-bit discrimination value of the NTQ106. The discrimination value stored for the HCS300/301 includes the overflow bits.

The second portion of validation involves checking **synchronization** information for that particular encoder. The synchronization counter transmitted by all encoders is 16 bits long. Two copies of the full synchronization counter are stored for all valid encoders. The storing of two copies of the synchronization information protects the decoder from loosing synchronization with an encoder if one of the counters is corrupted.

FIGURE 3: PINOUTS OF MICROCHIP KEELOQ DECODER



FUNCTIONAL INPUTS AND OUTPUTS

TABLE 5: MICROCHIP DECODER FUNCTIONAL INPUTS AND OUTPUTS

Mnemonic	Pin Number	Input / Output	Function
RF IN	18	I	Demodulated PWM signal from RF receiver. The decoder uses this input to receive encoder transmissions.
LEARN INIT	1	I	Input to initiate learning.
LEARN INDICATION	2	O	Output to show the status of the learn process (in an integrated system this will be combined with the system status indicator).
FUNC OK	13	O	Indication that the received button code matches the learned button code.
S0, S1, S2, S3	6, 7, 8, 9	O	Function outputs, correspond to encoder input pins.
EE DIO	10	I/O	EEPROM Data.
EE CK	11	O	EEPROM Clock.
EE CS	12	I	EEPROM Chip Select.

© 1997 Microchip Technology Inc.

PROGRAM FLOW

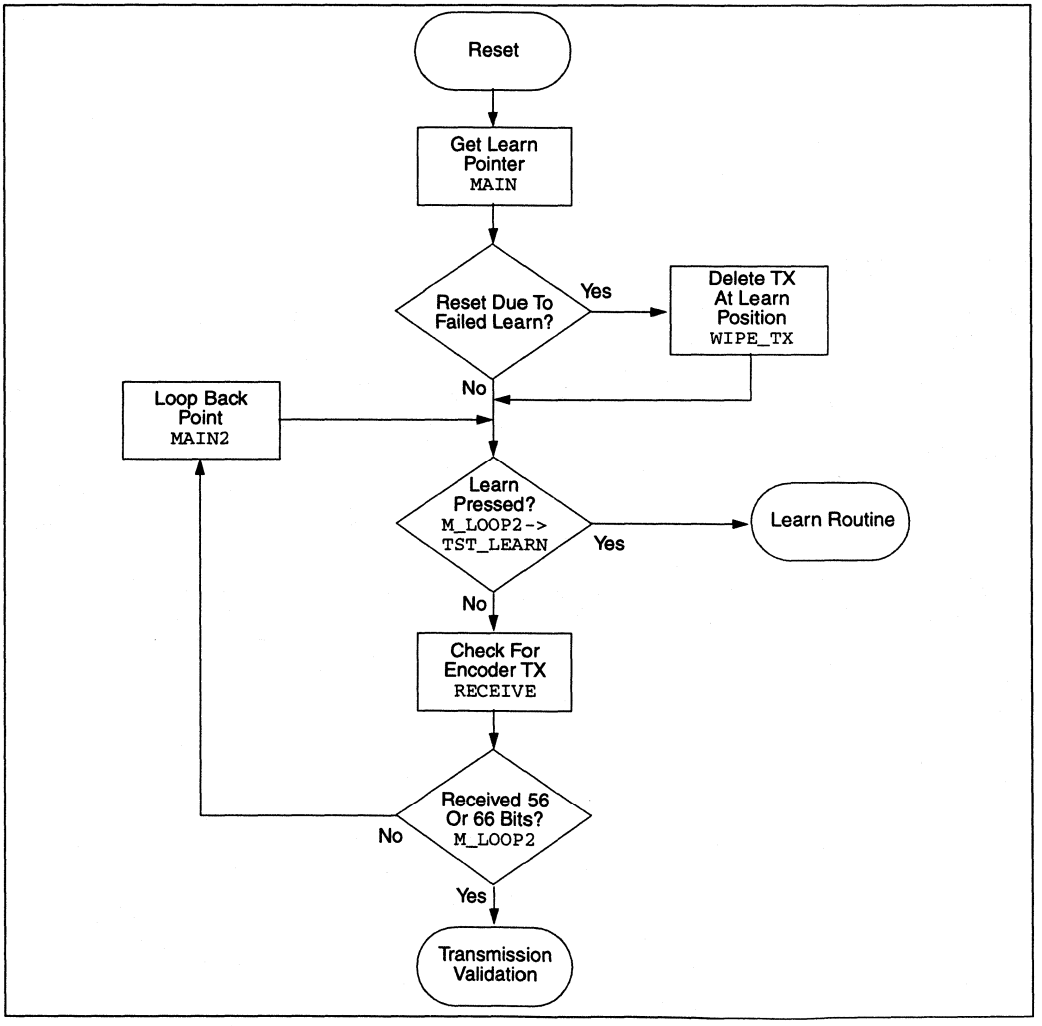
The software for the Microchip decoder has been written for the PIC16C56 microcontroller. The compiler used is MPASM. The operating frequency of the PIC16C56 is 4 MHz. The clock speed is important as the reception routine (RECEIVE) has some critical timing specifications. Other decoder functions that rely on a 4 MHz clock speed are the hold times of the various outputs, time-outs, etc.

The main program flow is described here. More detailed descriptions of the modules can be found further in the application note. On power-up the decoder reads the learn pointer from the external EEPROM. The status flags are checked to see if a learn routine was interrupted when the microcontroller was reset. If so, it is assumed the learn cycle was not successfully completed and the encoder at the learn pointer subsequently deleted (WIPE_TX).

The encoder then enters the main loop where it spends most of its time. The main loop checks to see if the learn button is being activated (TST_LEARN). If so, the decoder enters the learn mode.

If learn has not been initiated, the microcontroller then checks for transmissions from encoders (RECEIVE). If either 56 bits (NTQ106) or 66 bits (HCS encoders) are received, the microcontroller validates the transmission received. If the transmission received is a valid transmission from an encoder learned into the system, the system sets the appropriate outputs (M_BUT).

FIGURE 4: MICROCHIP DECODER MAIN PROGRAM FLOW



© 1997 Microchip Technology Inc.

FUNCTIONAL MODULES

Reception

The reception routine (RECEIVE) is based on a reliable algorithm which has successfully been used in previous implementations of KEELOQ decoders. Automatic baud rate detection is used to compensate for variations in baud rate of different encoders of a specific type, as well as the difference in baud rate between different encoders (NTQ106, HCS200, and HCS300). The reception routine is able to handle 56- and 66-bit transmissions. The reception routine is able to determine the type of encoder by the number of bits in the transmission.

The reception algorithm performs the following functions when an output is detected from the receiver:

1. Calibrate on the header low period to determine the actual elemental period for the transmission being received. The required elemental period is 10% of the low header period. In the diagram below (Figure 5) the header calibration sample points are marked 1 through 3. The calibration flow chart (Figure 6) shows at what points in the source code samples 1, 2, and 3 are taken.
   Elemental periods outside the capture range of the algorithm (either too long or too short) should be rejected, since they would be due either to noise or to reception of an incomplete signal.
2. Using the determined elemental period, three samples after the first rising edge following the header are taken. The first sample is taken half an elemental period after the rising edge (Sample 4); the second, one elemental period later (Sample 5), and the third, another one elemental period later (Sample 6). The first sample must be high, the second could be either high or low, and the third sample must be low. If either the first or the third sample is not as expected, the attempt at capturing a transmission is abandoned. In the diagram below (Figure 5), the data sample points are points 4 through 6. The flow chart describing data reception (Figure 7) shows where in the code the samples are taken.
3. If all 56 or 66 bits have been captured, each with the correct first and third elements, the transmission can be assumed to be correct, and decryption can commence.

The receiving routine should be called often enough to ensure that the high portion in the header is not missed (Sample 1, Figure 5).

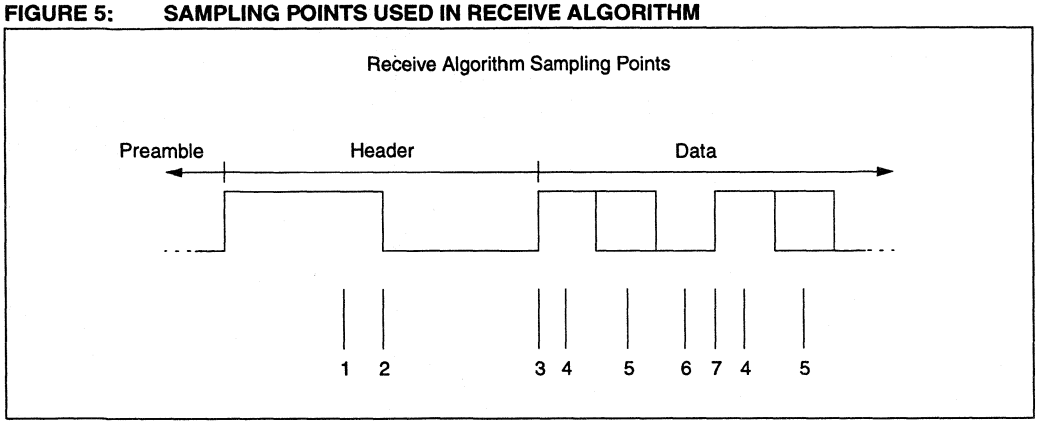
In systems where the receive routine is called to check if there is activity on the receiver input, the routine should poll the input for a valid transmission for at least the time taken to complete one transmission if activity is detected on the input line. This makes provision for the receive routine being called while a transmission is in progress. Having missed the first header, the first transmission will be invalid and be discarded. The decoder should continue sampling the input through the guard time in order to catch the next header and transmission (i.e., for a decoder designed to capture HCS300 transmissions the time spent polling for a valid transmission should be at least 100 ms if activity is detected in the input line).

The diagram below (Figure 5) gives all the major sampling points in the receive algorithm.

Note: The sample points are labeled in the receive routine flow diagrams that follow.

6

FIGURE 5: SAMPLING POINTS USED IN RECEIVE ALGORITHM



© 1997 Microchip Technology Inc.

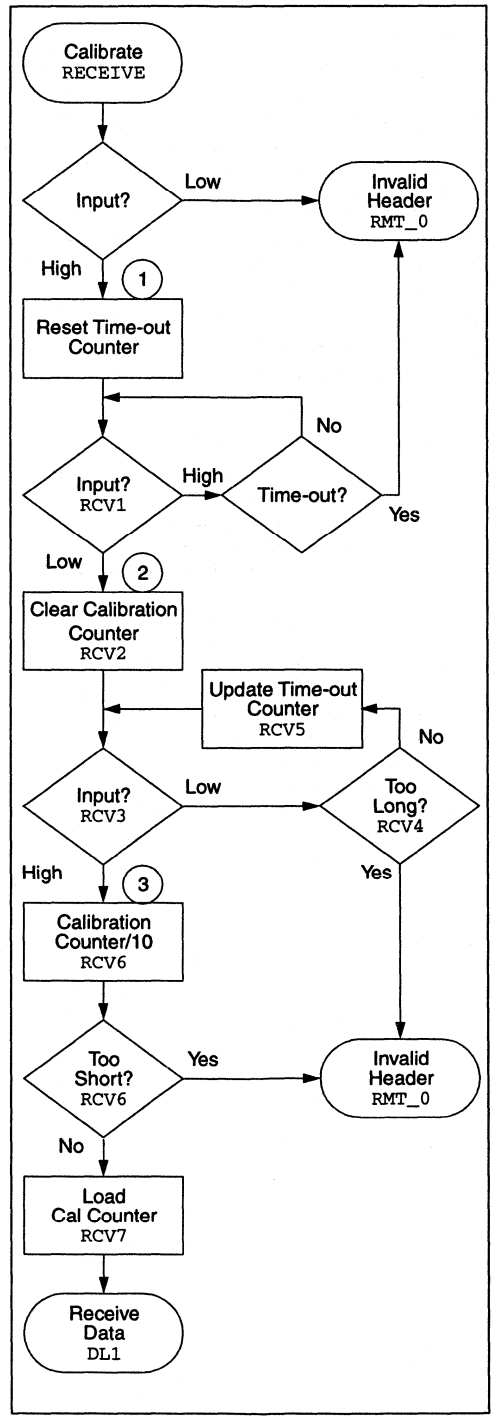
Flow Diagrams

The first flow diagram (Figure 6) describes the calibration routine which is used to determine the actual transmission rate of the encoder so that the decoder can compensate for deviations from nominal timing. There are four different exit points, each of which should branch to a point in the program where housekeeping and input monitoring can be resumed. There is only one exit point for a valid calibration operation (RCV7). At this point, it is assumed that a valid header has been received and that a string of data bits will follow.

The second flow diagram (Figure 7) handles the reception of bits once the calibration routine has been successfully completed. The data bits are all sampled three times each to ensure that a noise free transmission has been received. The receive routine uses the calibrated elemental period, determined in the calibration routine, to ensure that the samples are spaced correctly. The routine resynchronizes itself on the rising flank of each bit. Provision for identification of the encoder type, based on the number of bits received, is included. A 56-bit transmission emanates from a NTQ106, while a 66-bit transmission comes from a Microchip HCSXXX encoder. The last two bits of the HCS300 transmission, VLOW and Repeat, are ignored by the Microchip decoder.

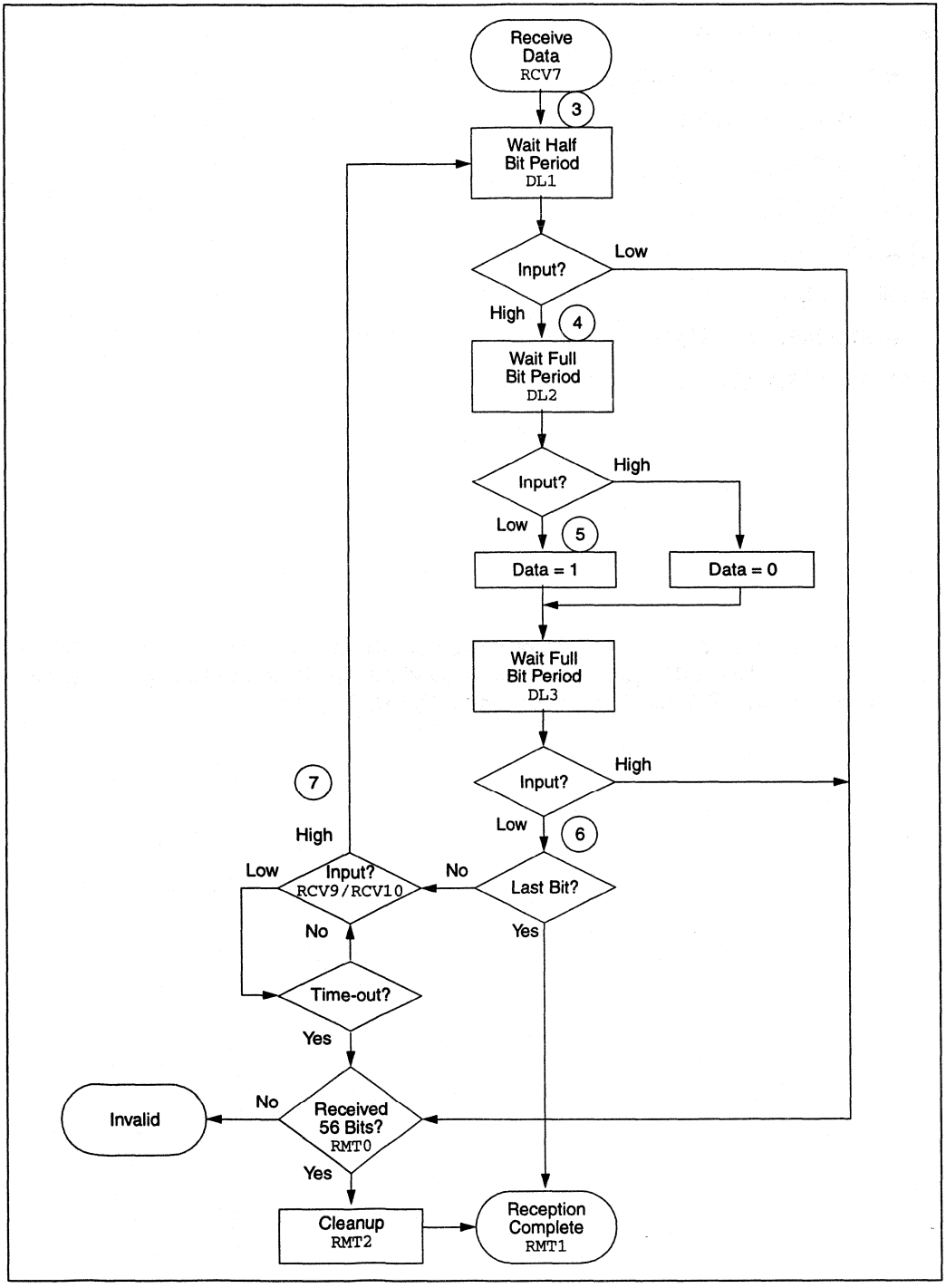
If all of the control samples in all of the bits are sampled correctly (i.e., the first element is high and the last element is low), the routine checks whether 56 bits have been received correctly (NTQ106 transmission). If not, the routine returns to the calling procedure.

FIGURE 6: CALIBRATION FLOW CHART



© 1997 Microchip Technology Inc.

FIGURE 7: DATA RECEPTION IN RECEIVE



© 1997 Microchip Technology Inc.

VALIDATION

SYNCHRONIZATION

FUNCTION INTERPRETATION

OUTPUT ACTIVATION

KEY GENERATION

DECRYPTION

LEARN

ROM MEMORY MAP

EEPROM MEMORY MAP

RAM MEMORY MAP

The confidential and proprietary information contained in this section of AN642 has been removed. The full application note is available under a license agreement and can be ordered as DS40149 from Microchip Technology Inc.

© 1997 Microchip Technology Inc.

DEVICE PINOUTS

The device used in the application note is a PIC16C56 PDIP.

TABLE 6: DEVICE PINOUTS

PIN	PIC16C56 Function	Decoder Function	PIN	PIC16C56 Function	Decoder Function
1	PORTA Bit2	LEARN Input Active Low	18	PORTA Bit1	RF Input
2	PORTA Bit3	LRN IND Output Active High	17	PORTA Bit0	Not used
3	TIME	Connect to VDD	16	Osc In	RC osc (4 MHz)
4	$\overline{MCLR}$	Brown out detect	15	Osc Out	
5	GND	Ground	14	VDD	+5V supply
6	PORTB Bit0	S0	13	PORTB Bit7	FUNC OK
7	PORTB Bit1	S1	12	PORTB Bit6	EEPROM CS (1)
8	PORTB Bit2	S2	11	PORTB Bit5	EEPROM CLK (2)
9	PORTB Bit3	S3	10	PORTB Bit4	EEPROM DIO (3+4)

TIMING PARAMETERS

TABLE 7: TIMING PARAMETERS

Parameter	Typical	Unit
Output activation duration	524	ms
Output pause if new function code received	131	ms
Erase all duration	8.4	s
Learn mode time-out	33.6	s
Learn successful $\overline{LED}$ flash duration	4.2	s
Learn successful $\overline{LED}$ flash rate	3.8	Hz
Learn failure $\overline{LED}$ on duration	1	s

SOURCE CODE LISTING

A diskette is supplied containing source code for the Microchip decoder in the file `mcdec12.asm`. The code has been compiled using MPASM v01.30.01. Certain functions are dependent on the oscillator speed for correct functioning. Examples of time dependent functions include `RECEIVE` and `TST_RTCC`. The PIC16C56 Microcontroller should run at 4 MHz.

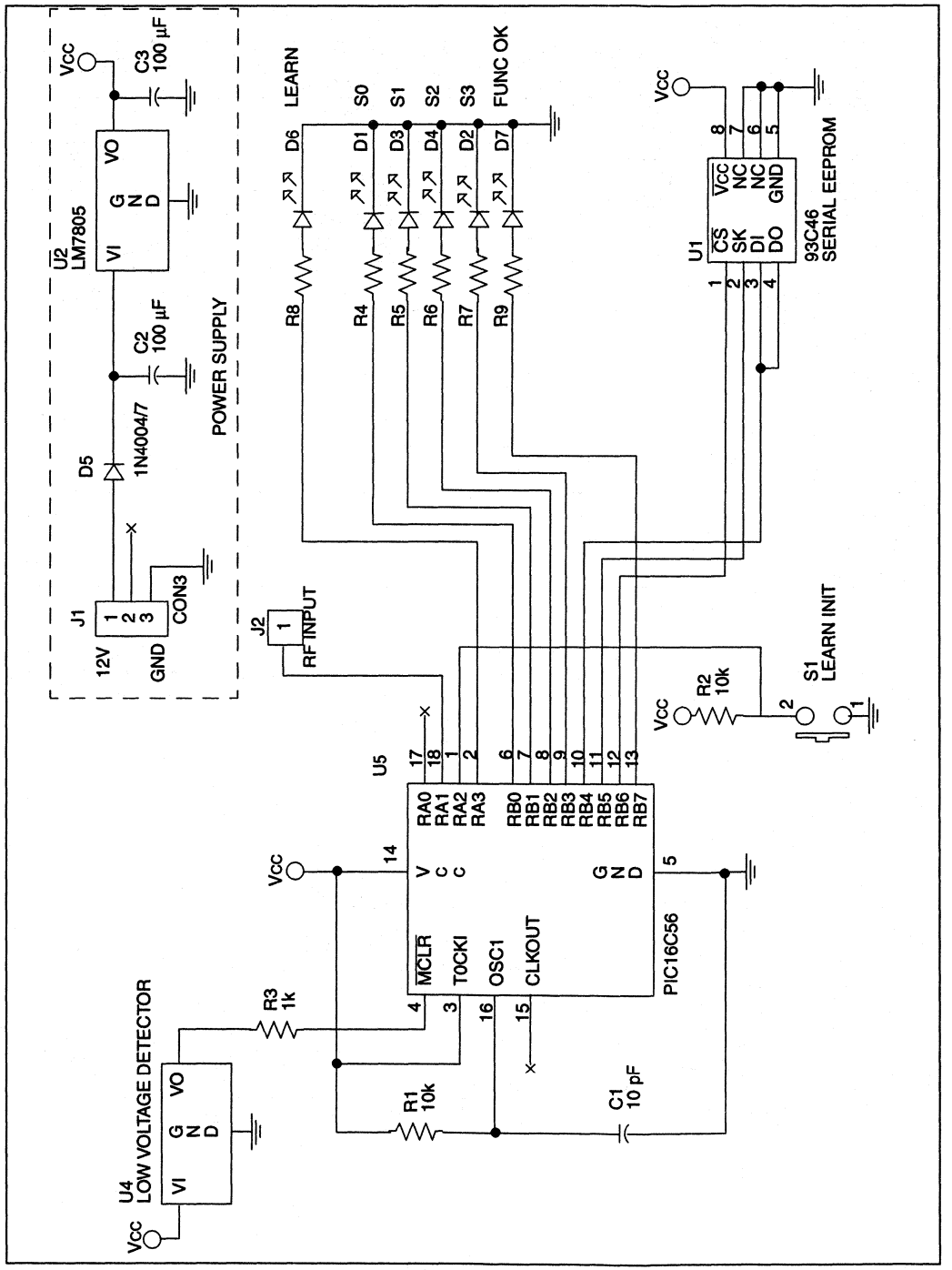
LIST OF IMPORTANT FUNCTIONS

Function Name	Description	Line Number	Function Length
CALC_KEY	Key generation routine.	1219	62
DECRYPT	Decryption routine for Code Hop.	469	61
EEREAD	The data in the EEPROM at ADDRESS is read and decrypted to TMP1 and TMP2. Note that TMP1, TMP2 and ADDRESS are user defined registers.	945	31
EEWRITE	The data in TMP1 and TMP2 is encrypted and written to the EEPROM at ADDRESS. Note that TMP1, TMP2 and ADDRESS are user defined registers.	860	48
M_DIS	Check discrimination value.	1605	19
M_CNT	Check synchronization (counter) values.	1633	69
RECEIVE	Start of the RF receive routine.	580	121
TST_LEARN1	Check for learn mode and entry to learn.	775	21
TST_RTCC	Check TIMER0 and do whatever real time tasks are required.	397	38

© 1997 Microchip Technology Inc.

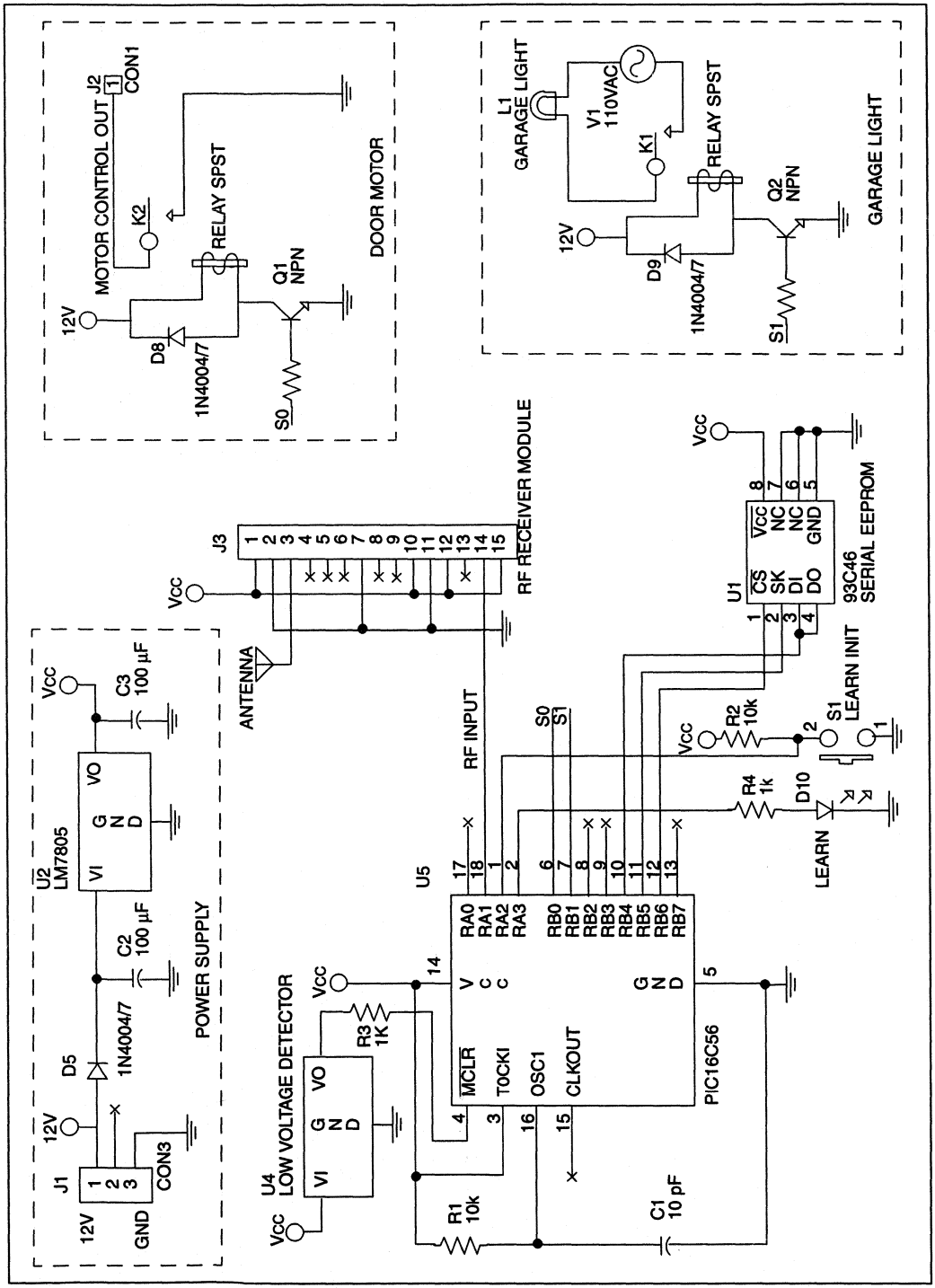
APPENDIX A: APPENDIX SCHEMATIC DIAGRAMS

FIGURE A1: SCHEMATIC DIAGRAM OF MICROCHIP KEELOQ DECODER



© 1997 Microchip Technology Inc.

FIGURE A2: TYPICAL GARAGE DOOR OPENER SCHEMATIC



NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN662

KEELOQ® Code Hopping Decoder Using Secure Learn

Author: Steven Dawson
Microchip Technology Inc.

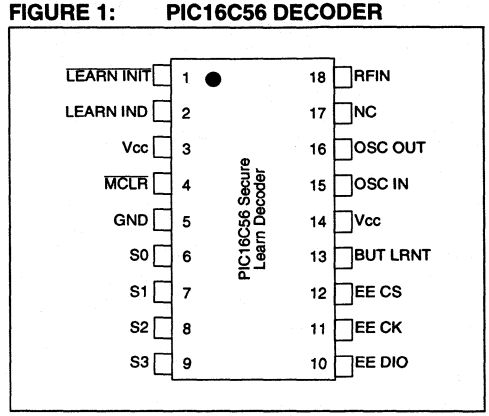
OVERVIEW

This application note fully describes the working of a code hopping decoder implemented on a Microchip PIC16C56 microcontroller. This decoder uses the secure learn (seed-based) method of learning new transmitters. This application note describes the various KEELOQ code hopping encoders that can be used with the decoder, the decoder hardware, and the various software modules comprising the system. The software can be used to implement a stand alone decoder or integrate with full function security systems. The decoder supports the Microchip HCS200, HCS300, HCS301, HCS360, and HCS361 KEELOQ code hopping encoders.

KEY FEATURES

- Stand alone decoder
- Compatible with Microchip HCS200, HCS300, HCS301, HCS360, and HCS361 encoders
- Automatic bit rate detection
- Automatic encoder type detection
- Four function outputs
- Six learnable transmitters
- RC Oscillator

FIGURE 1: PIC16C56 DECODER



Notice:

This is a non-restricted version of Application Note AN652 which is available under the KEELOQ License Agreement. The license agreement can be ordered from the Microchip Literature Center as DS40149.

© 1997 Microchip Technology Inc.

INTRODUCTION TO KEELOQ ENCODERS

All KEELOQ encoders use the KEELOQ code hopping technology to make each transmission by an encoder unique. The encoder transmissions have two parts. The first part changes each time the encoder is activated and is called the hopping code part and is encrypted. The second part is the unencrypted part of the transmission, principally containing the encoder's serial number identifying it to a decoder. Refer to DS91002, *Introduction to KEELOQ*.

Hopping Code

The hopping code contains function information, a discrimination value, and a synchronization counter. This information is encrypted by an encryption algorithm before being transmitted. A 64-bit encryption key is used by the encryption algorithm. If one bit in the data that is encrypted changes, the result is that an average of half the bits in the output will change. As a result, the hopping code changes dramatically for each transmission and can not be predicted.

Function Information

The encoder transmits up to four bits of function information. Up to 15 different functions are available.

Discrimination Value

Stored in the encoder EEPROM, this information can be used to check integrity of decryption operation by a decoder. If known information is inserted into the transmitted string before encryption, the same information can be used at the decoder to check whether the information has been decrypted correctly. 12 bits (including overflow bits) are available in the Microchip HCSXXX encoders.

Synchronization Counters

The transmitted word contains a 16-bit synchronization counter. The synchronization information is used at the decoder to determine whether a transmission is valid or is a repetition of a previous transmission. Previous codes are rejected to safeguard against code grabbers. The HSC300 and HCS301 encoders transmit two overflow bits which may be used to extend the range of the synchronization counter from 65,536 to 196,608 button operations. The HCS360 and HCS361 encoders transmit one overflow bit which can be used to extend the range of the synchronization counter from 65,536 to 131,071 button operations.

Unencrypted Code

Serial Number

The encoder's serial number is transmitted every time the button is pressed. The serial number is transmitted unencrypted as part of the transmission and serves to identify the encoder to the decoder.

Other Status and Function Information

The HCS200, HCS300, and HCS301 encoders include provision for four bits of function information and two status bits in the fixed code portion of its transmission. The two status bits indicate whether a repeated transmission is being sent, and whether the battery voltage is low. The HCS200 does not send repeated transmission information, and the bit is permanently set to '0'.

The HCS360/361 encoders transmit two bits that are used as a Cyclic Redundancy check. These bits can be used to check the integrity of the reception. Additionally, the HCS360 and HCS361 encoders can extend the length of the serial number from 28 bits to 32 bits, replacing the unencrypted function code.

Seed Transmissions

The Microchip HCSXXX encoders all have the ability to transmit a fixed seed. The seed value is programmed into the encoder when the encoder is first initialized along with the counters, key, serial number, and other information. The seed length differs from encoder to encoder with the HCS200, HCS300, and HCS301 having a 32-bit seed. The HCS360 and HCS361 encoders have a 48-bit seed. The HCS200, HCS300, and HCS301 encoders transmit the seed if all the inputs are activated simultaneously (S0, S1, S2, and S3). The HCS360 and HCS361 encoders transmit the seed immediately if S0 and S3 are activated or delayed if S0 and S1 are activated for more than 3 seconds.

Table 1 summarizes seed transmissions in the Microchip HCSXXX encoders.

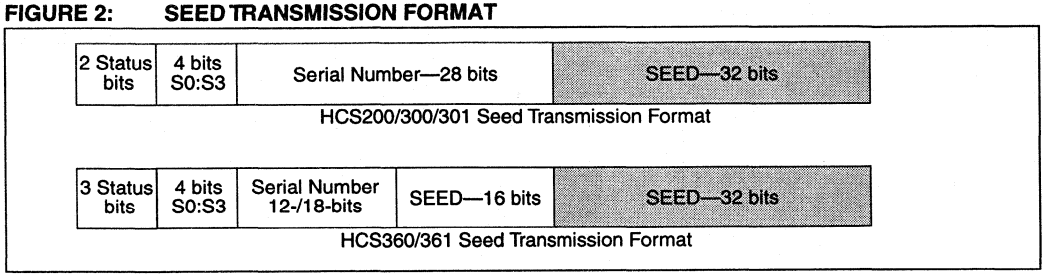
© 1997 Microchip Technology Inc.

TABLE 1: HCSXXX ENCODER SEED TRANSMISSION SUMMARY

Encoder	Seed Length	Seed Transmission Activation
HCS200	32 bits	Seed transmitted immediately if S0, S1, and S2 are activated.
HCS300	32 bits	Seed transmitted immediately if S0, S1, S2, and S3 are activated.
HCS301	32 bits	Seed transmitted immediately if S0, S1, S2, and S3 are activated.
HCS360	48 bits	Seed transmitted immediately if S0 and S3 are activated. Seed transmitted after 3 seconds if S0 and S1 are activated.
HCS361	48 bits	Seed transmitted immediately if S0 and S3 are activated. Seed transmitted after 3 seconds if S0 and S1 are activated.

The seed transmission is transmitted in place of the HOP transmission and is not encrypted. When the seed is transmitted, the transmitted word changes as shown in Figure 2 below.

FIGURE 2: SEED TRANSMISSION FORMAT



6

Transmission Format Summary

Table 2 contains a summary of the information contained in transmissions from each of the KEELOQ encoders that can be learned by the Microchip decoder.

FIGURE 3: DECODER BLOCK DIAGRAM

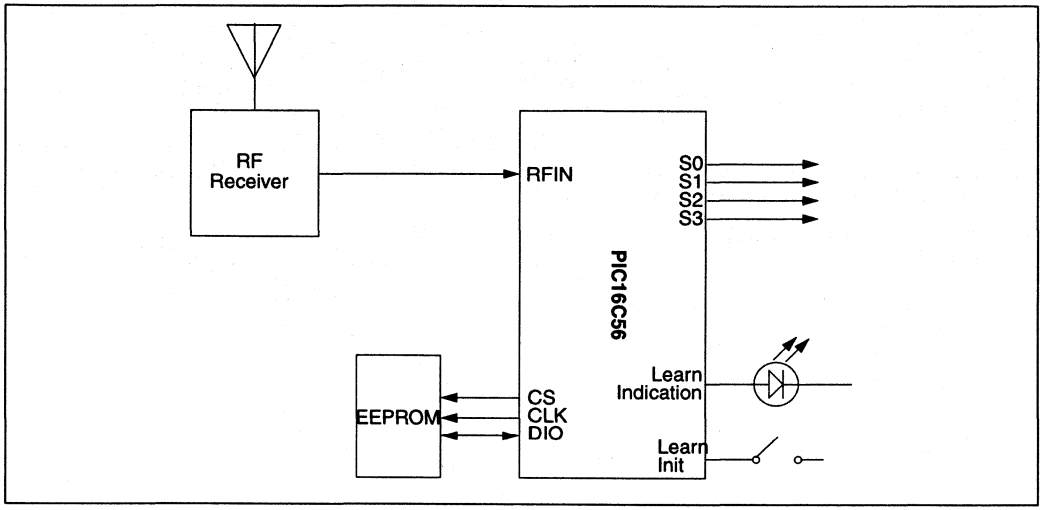


TABLE 2: KEELOQ ENCODER TRANSMISSION SUMMARY

	HCS200[1]	HCS300/301	HCS360/361
Total transmission length	66 bits	66 bits	67 bits
Hopping code portion (total length)	32 bits	32 bits	32 bits
Function bits	3 bits (+1[1])	4 bits	4 bits
Discrimination bits	12 bits	12 bits	12 bits
Synchronization	16 bits	16 bits	16 bits
Fixed portion (total length)	34 bits	34 bits	35 bits
Serial number	28 bits	28 bits	28/32 bits[2]
Function bits	3 bits (+1[1])	4 bits	4/0 bits[2]
Status bits			
VLOW	1 bit	1 bit	1 bit
RPT	0	1 bit	0
CRC	0	0	2 bits

[1]The HCS200 transmissions are padded to retain compatibility with the HCS300/301 encoders.
[2]User selectable with a 32-bit serial number or a 28-bit serial number, and 4 unencrypted function bits.

The code hopping portion is always transmitted first (LSB to MSB), followed by the fixed portion (LSB to MSB).

TABLE 3: HCS200† AND HCS300/301 TRANSMISSION FORMAT

Hop Code			Serial Number			Function	Status
LSB	32 bits	MSB	LSB	28 bits	MSB	4 bits	2 bits

†The HCS200 transmissions are padded to retain compatibility with the HCS300/301 encoders.

TABLE 4: HCS360/361 TRANSMISSION FORMAT

Hop Code			Serial Number			Function	Status
LSB	32 bits	MSB	LSB	28/32 bits	MSB	4/0 bits	3 bits

Bits are transmitted from left to right (i.e., the code hopping part first).

© 1997 Microchip Technology Inc.

PWM Format

In general, all KEELOQ encoders share a common transmission format:

A **preamble** to improve biasing of decision thresholds in super-regenerative receivers. The preamble consists of alternate on and off periods, each lasting as long as a single elemental period.

A **calibration** header consisting of a low period of 10 elemental periods. Calibration actions should be performed on the low period of the header to ensure correct operation with header chopping.

A **string** of 66 or 67 pulse-width modulated bits, each consisting of three elements. The first element is high, the second contains the data transmitted and is either high or low, the third element is always low.

A **guard** period is usually left between the transmissions. During this period nothing is transmitted by the encoder.

Figure 4 shows the sampling points when sampling the data bits. The first and last elements are used exclusively to verify the integrity of the received symbol. The first element (sample point A) is always high, the second (sample point B) is the complement of the data bit being sent, and the final element (sample point C) is always low. Because the period between the low portion of a bit (sample point C) and the rising edge of the following bit (sample point X) can vary, the rising edge of the first element (sample point X) is used to resynchronize the receiving routine to each incoming bit.

If random noise is being received, the probability of a set of three samples producing a valid combination is only $2^{-2} = 1/4$. For a string of 66 bits, the corresponding figure is $2^{-132} < 2 \times 10^{-40}$.

Integrity checking on incoming signals is important. Code hopping signals require significant processing, as well as EEPROM access, to decrypt. Unnecessary processing can be avoided by not attempting to decrypt incoming codes that have bit errors.

FIGURE 4: KEELOQ PWM TRANSMISSION FORMAT

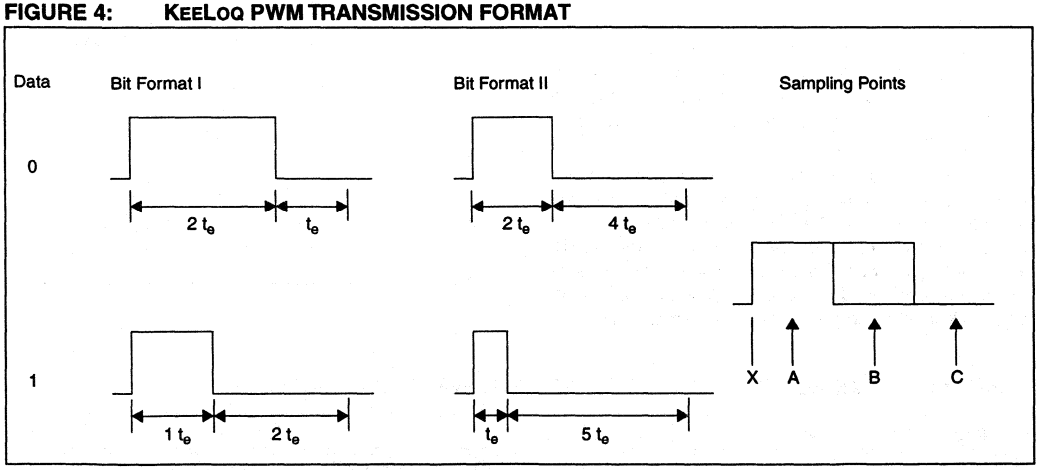


TABLE 5: TRANSMISSION FORMATS

Encoder	Preamble	Header Chopping	PWM Bit Format		Hopping Code Length	Unencrypted Code Length
			I	II		
HCS200	•	•	•	—	32 bits	34 bits
HCS300/301	•	•	•	—	32 bits	34 bits
HCS360	•	•	•	—	32 bits	35 bits
HCS361	•	•	•	•	32 bits	35 bits

Microwire is registered trademark of Motorola.

© 1997 Microchip Technology Inc.

DECODER IMPLEMENTATION

The Microchip decoder's primary hardware components are a PIC16C56 RISC microcontroller and a 93C46 EEPROM as shown in the decoder schematic in Figure 11. However, this solution can be implemented in any PICmicro™ microcontroller with at least 1K words of program memory. The operating frequency of the controller is 4 MHz. The microcontroller is used to capture transmissions from the various encoders, decrypt transmissions captured, and check the validity of the transmission based on the information in the decrypted transmission and information stored in the EEPROM. If a transmission from a valid encoder is received, the decoder activates the outputs dictated by the transmission.

Encoder information, such as serial number, synchronization information, and decryption key is stored externally in an EEPROM. The EEPROM used is a Microchip 93C46 Microwire® Serial EEPROM. The information stored in the EEPROM is encrypted to protect the contents. The EEPROM encryption is less secure than the KEELOQ code hopping algorithm.

As can be seen from the section on encoder transmissions, there are differences in the transmission formats of the different encoders that can be used with the decoder. The following section summarizes how the differences in transmitted data are dealt with by the decoder.

As the serial number information follows after the code hopping portion of the transmission, any number of serial number bits can be received and processed. In the Microchip decoder described, the complete serial number (28 bits) is stored.

The serial number is used to identify the memory block used to store the 64-bit decryption key for a particular encoder because of the relationship between serial number, seed, and the decryption key. In other words, the serial number is stored with the key. When a transmission is received, the decoder finds the correct memory block by checking all blocks until a matching serial number is found. The key is then retrieved from that particular memory block. A serial number of 0000000_{16} is considered invalid and is ignored by the decoder.

After matching the received and stored serial number, validation of a received transmission consists of two steps. The first includes checking the integrity of the decryption operation. Here the decoder compares the 12-bit discrimination value received with the stored discrimination value. The discrimination value stored with the HCS300/301/360/361 includes overflow bits and user bits.

The second portion of validation involves checking synchronization information for that particular encoder. The synchronization counter transmitted by all encoders is 16 bits long. Two copies of the full synchronization counter are stored for all valid encoders. The storing of two copies of the synchronization information protects the decoder from loosing synchronization with an encoder if one of the counters is corrupted.

PINOUTS OF MICROCHIP KEELOQ DECODER

FIGURE 5: FUNCTIONAL INPUTS AND OUTPUTS

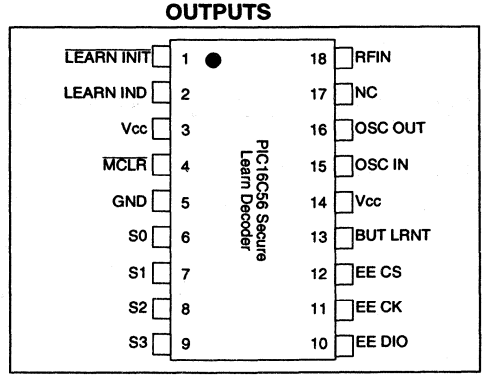


TABLE 6: MICROCHIP DECODER FUNCTIONAL INPUTS AND OUTPUTS

Mnemonic	Pin Number	Input / Output	Function
RF IN	18	I	Demodulated PWM signal from RF receiver. The decoder uses this input to receive encoder transmissions.
LEARN INIT	1	I	Input to initiate learning, active low.
LEARN IND	2	O	Output to show the status of the learn process (in an integrated system this will be combined with the system status indicator).
BUT LRNT	13	O	Indication that the received function code matches the learned function code.
S0, S1, S2, S3	6, 7, 8, 9	O	Function outputs—corresponds to encoder input pins.

Microwire is registered trademark of Motorola.

© 1997 Microchip Technology Inc.

PROGRAM FLOW

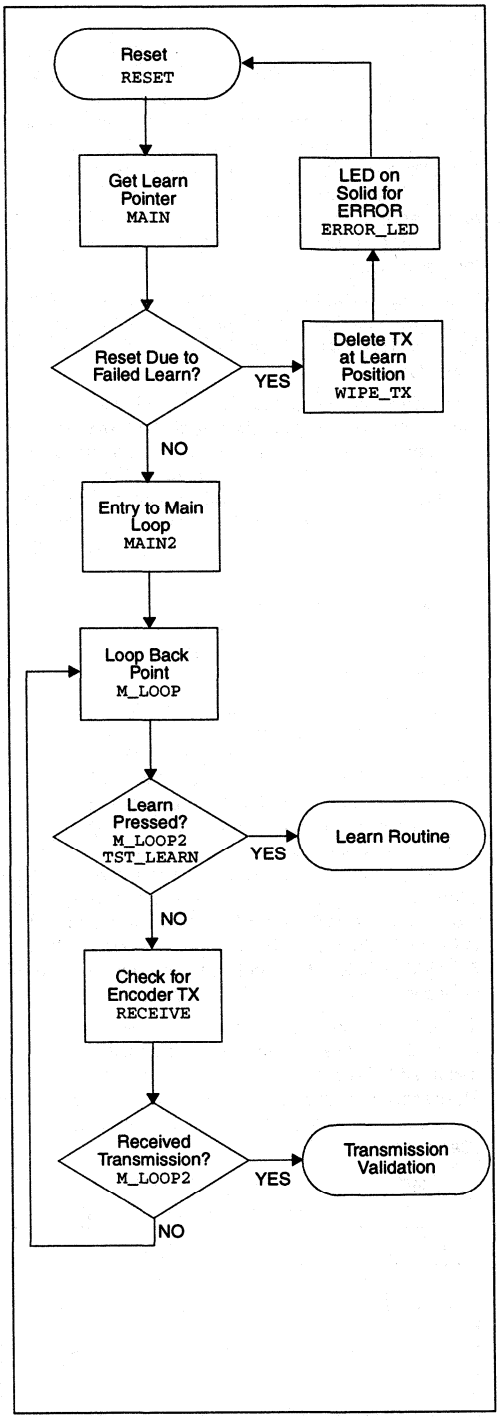
The software for the Microchip decoder has been written for the PIC16C56 microcontroller. The compiler used is MPASM version 01.30.01. The operating frequency of the PIC16C56 is 4 MHz. The clock speed should be kept as close as possible to 4 MHz as the reception routine (RECEIVE) is dependent on the 4 MHz clock for correct functioning. Other decoder functions that rely on a 4 MHz clock speed are the hold times of the various outputs and time-outs. The main program flow is described here. Detailed descriptions of individual functions can be found further in the application note.

As can be seen from Figure 6, the decoder reads the learn pointer, which stores the next position to be overwritten when the decoder enters a learn sequence and decoder status flags, from the external EEPROM on power-up. The status flags are checked to see if a learn routine was interrupted when the microcontroller was reset. If so, it is assumed the learn cycle was not successfully completed, and the encoder at the learn pointer is subsequently deleted (WIPE_TX).

The encoder then enters the main loop where it spends most of its time. The main loop checks to see if the learn button is being activated (TST_LEARN). If so, the decoder enters the learn mode described in the "Learn" section in AN652.

If learn has not been initiated, the microcontroller then checks for transmissions from encoders (RECEIVE) as described on page 40. If a transmission from an encoder has successfully been received, the microcontroller validates the transmission received as described in the "Transmission Validation" in AN652. If the transmission received is a valid transmission from an encoder learned into the system, the system sets the appropriate outputs (M_BUT).

FIGURE 6: MICROCHIP DECODER MAIN PROGRAM FLOW



6

© 1997 Microchip Technology Inc.

FUNCTIONAL MODULES

Reception

The reception routine (RECEIVE) is based on a reliable algorithm which has successfully been used in previous implementations of KEELOQ decoders. Automatic bit-rate detection is used to compensate for variations in bit rate of different encoders of a specific type, as well as the differences in bit rate between different encoders (HCS200, HCS300, and HCS360). The reception routine is able to receive 66-bit transmissions. The reception routine is able to determine the number of bits in the transmission.

The reception algorithm performs a number of functions when an output is detected from the receiver. Figure 7 gives all the major sampling points in the reception algorithm.

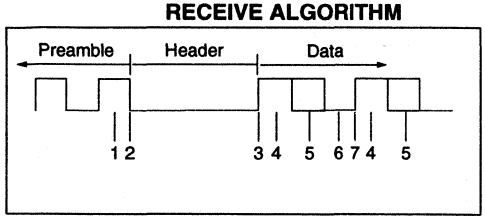
The reception algorithm calibrates on the low period of the header to determine the actual elemental period for the transmission being received. The required elemental period is 10% of the low header period. In Figure 7 the header calibration sample points are marked 1 through 3. The calibration flow chart (Figure 8) shows at what points in the program samples 1, 2, and 3 are taken.

Elemental periods outside the capture range of the algorithm (either too long or too short) are rejected, since they are due either to noise or to reception of an incomplete signal.

Using the determined elemental period, three samples after the first rising edge (Sample 3) following the header are taken. The first sample is taken half an elemental period after the rising edge (Sample 4); the second, one elemental period later (Sample 5), and the third, another one elemental period later (Sample 6). The first sample must be high, the second could be either high or low, and the third sample must be low. If either the first or the third sample is not as expected, the attempt at capturing a transmission is abandoned. In Figure 7, the data sample points are points 4 through 6. The flow chart describing data reception (Figure 9) shows where in the code the samples are taken.

If all 66 bits have been captured, each with the correct first and third elements, the transmission can be assumed to be correct, and decryption can commence. The receiving routine should be called often enough to ensure that the high portion in the header is not missed (Sample 1, Figure 7).

FIGURE 7: SAMPLING POINTS USED IN RECEIVE ALGORITHM



In systems where the reception routine is called to check if there is activity on the receiver input, the routine should poll the input for a valid transmission for at least the time taken to complete one transmission if activity is detected on the input line. This makes provision for the reception routine being called while a transmission is in progress. Having missed the first header, the first transmission will be invalid and be discarded. The decoder should continue sampling the input through the guard time in order to catch the next header and transmission (i.e., for a decoder designed to capture HCS300 transmissions the time spent polling for a valid transmission should be at least 100 ms if activity is detected in the input line).

Reception Algorithm Flow Chart

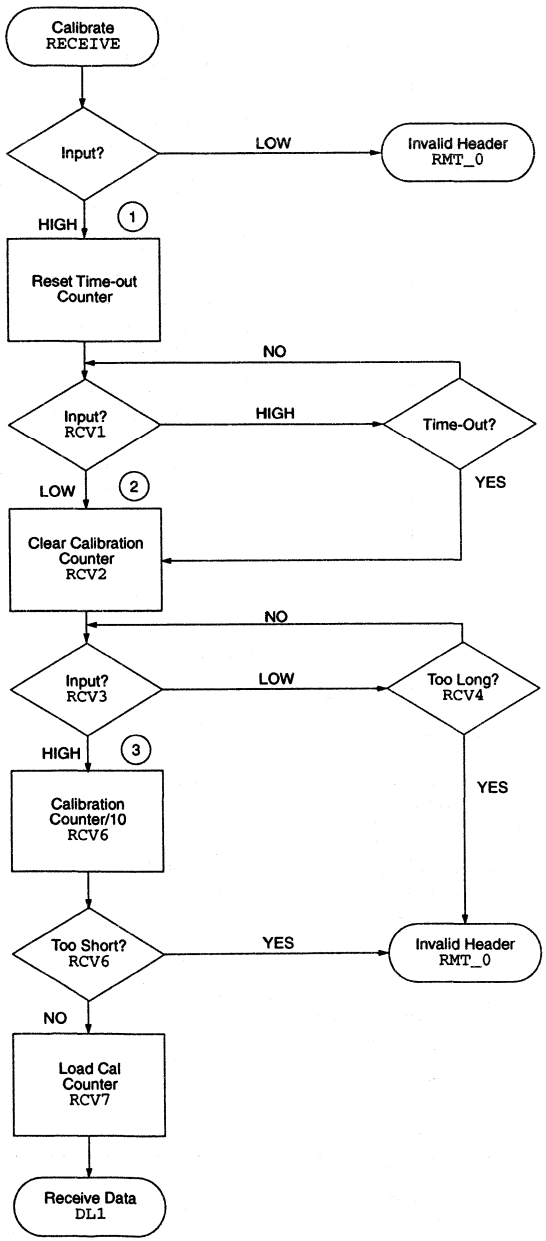
The first flow chart (Figure 8) describes the calibration routine which is used to determine the actual transmission rate of the encoder so that the decoder can compensate for deviations from nominal timing. There are four different exit points, each of which should branch to a point in the program where housekeeping and input monitoring can be resumed. There is only one exit point for a valid calibration operation (RCV7). At this point, it is assumed that a valid header has been received and that a string of data bits will follow.

The second flow chart (Figure 9) handles the reception of bits once the calibration routine has been successfully completed. The data bits are all sampled three times each to ensure that a noise free transmission has been received. The reception routine uses the calibrated elemental period, determined in the calibration routine, to ensure that the samples are correctly spaced. The routine resynchronizes itself on the rising flank of each bit. Only 60 bits of the data received are used by the Microchip decoder described, the decoder ignores the unencrypted function code and the status bits.

If the control samples in a given bit are sampled correctly (i.e., the first element is high and the last element is low), the routine checks whether more than 56 bits have been received correctly. If not, the routine returns to the calling procedure.

© 1997 Microchip Technology Inc.

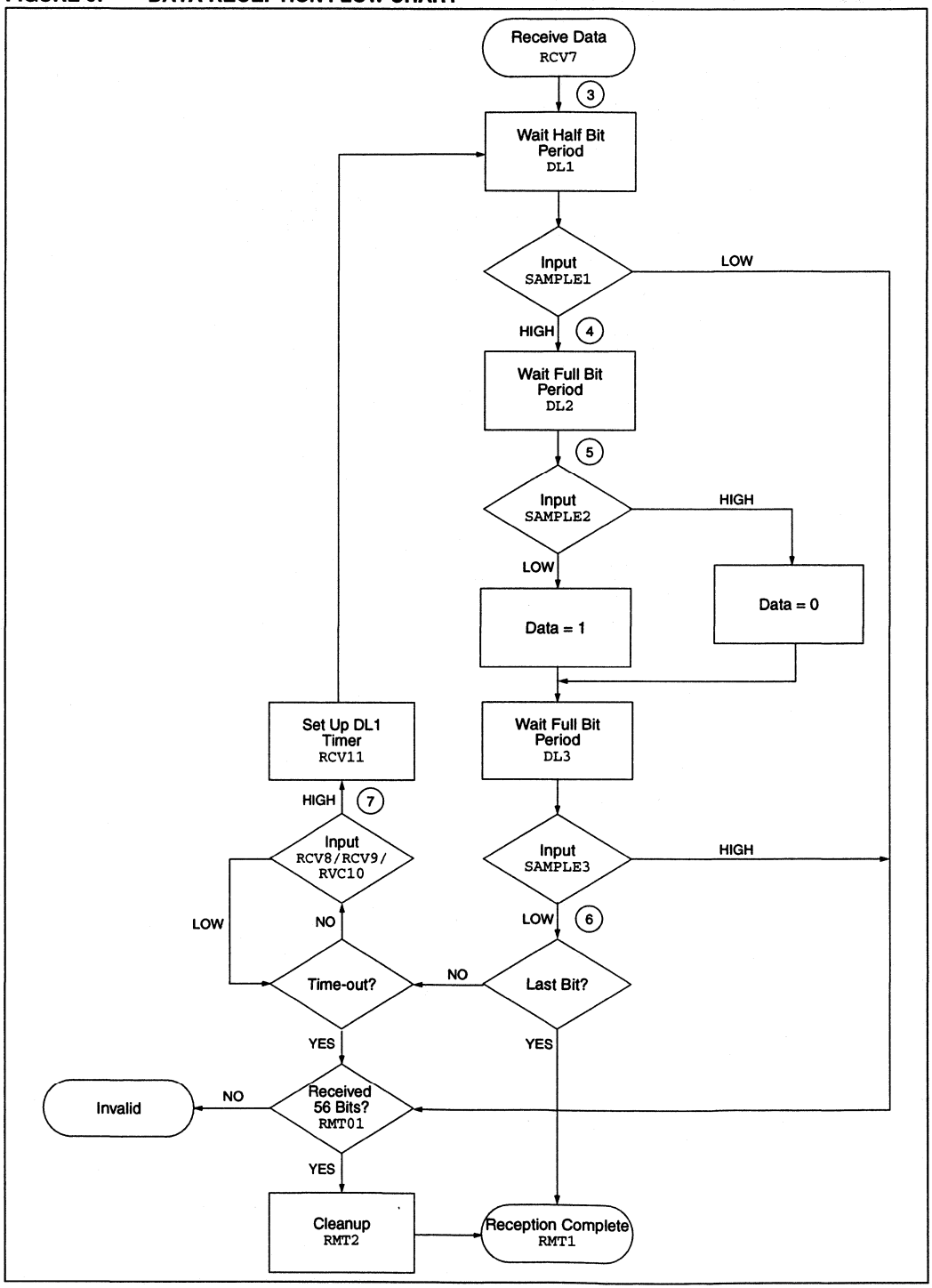
FIGURE 8: CALIBRATION FLOW CHART



6

© 1997 Microchip Technology Inc.

FIGURE 9: DATA RECEPTION FLOW CHART



© 1997 Microchip Technology Inc.

VALIDATION

SYNCHRONIZATION

FUNCTION INTERPRETATION

OUTPUT ACTIVATION

KEY GENERATION

DECRYPTION

LEARN

ROM MEMORY MAP

EEPROM MEMORY MAP

RAM MEMORY MAP

The confidential and proprietary information contained in this section of AN652 has been removed. The full application note is available under a license agreement and can be ordered as DS40149 from Microchip Technology Inc.

© 1997 Microchip Technology Inc.

DEVICE PINOUTS

The device used in the application note is a PIC16C56 PDIP.

TABLE 7: DEVICE PINOUTS

PIN	PIC16C56 Function	Decoder Function	PIN	PIC16C56 Function	Decoder Function
1	Port A Bit 2	LEARN Input	18	Port A Bit 1	RF Input
2	Port A Bit 3	LRN INDICATOR	17	Port A Bit 0	Not used
3	TIMER0	Connect to VDD	16	Osc In	RC osc (4 MHz)
4	$\overline{MCLR}$	Brown out detect	15	Osc Out	
5	GND	Ground	14	VDD	+5V supply
6	Port B Bit 0	S0	13	Port B Bit 7	BUT LRNT
7	Port B Bit 1	S1	12	Port B Bit 6	CS (93C46, pin 1))
8	Port B Bit 2	S2	11	Port B Bit 5	CLK (93C46, pin 2)
9	Port B Bit 3	S3	10	Port B Bit 4	DIO (93C46, pin 3 & 4)

TIMING PARAMETERS

TABLE 8: TIMING PARAMETERS

Parameter	Typical	Unit
Output activation duration	524	ms
Output pause if new function code received	131	ms
Erase all duration	8.4	s
Learn mode time-out	33.6	s
Learn successful LED flash duration	4.2	s
Learn successful LED flash rate	3.8	Hz
Learn failure LED on duration	1	s

SOURCE CODE LISTING

A diskette is supplied containing source code for the Microchip decoder in the file `MSLRN**.ASM`. The code has been compiled using MPASM v01.30.01. Certain functions are dependent on the oscillator speed for correct functioning. Examples of time dependent functions include `RECEIVE` and `TST_RTCC`. The PIC16C56 Microcontroller should run at 4 MHz.

TABLE 9: LIST OF IMPORTANT FUNCTIONS

Function Name	Description
CALC_KEY	Key generation routine.
DECRYPT	Decryption routine for Hop Code.
EEREAD	The data in the EEPROM at ADDRESS is read and decrypted to TMP1 and TMP2 (Note).
EEWRITE	The data in TMP1, and TMP2 is encrypted and written to the EEPROM at ADDRESS (Note).
M_DIS	Check discrimination value.
M_CNT	Check synchronization (counter) values.
RECEIVE	Start of the RF reception routine.
TST_LEARN1	Check for learn mode and entry to learn.
TST_RTCC	Check TIMER0 and do whatever real time tasks are required.

Note: TMP1, TMP2 and ADDRESS are user defined registers.

© 1997 Microchip Technology Inc.

APPENDIX A: SCHEMATIC DIAGRAMS

FIGURE A-1: SCHEMATIC DIAGRAM OF MICROCHIP KEELOQ DECODER

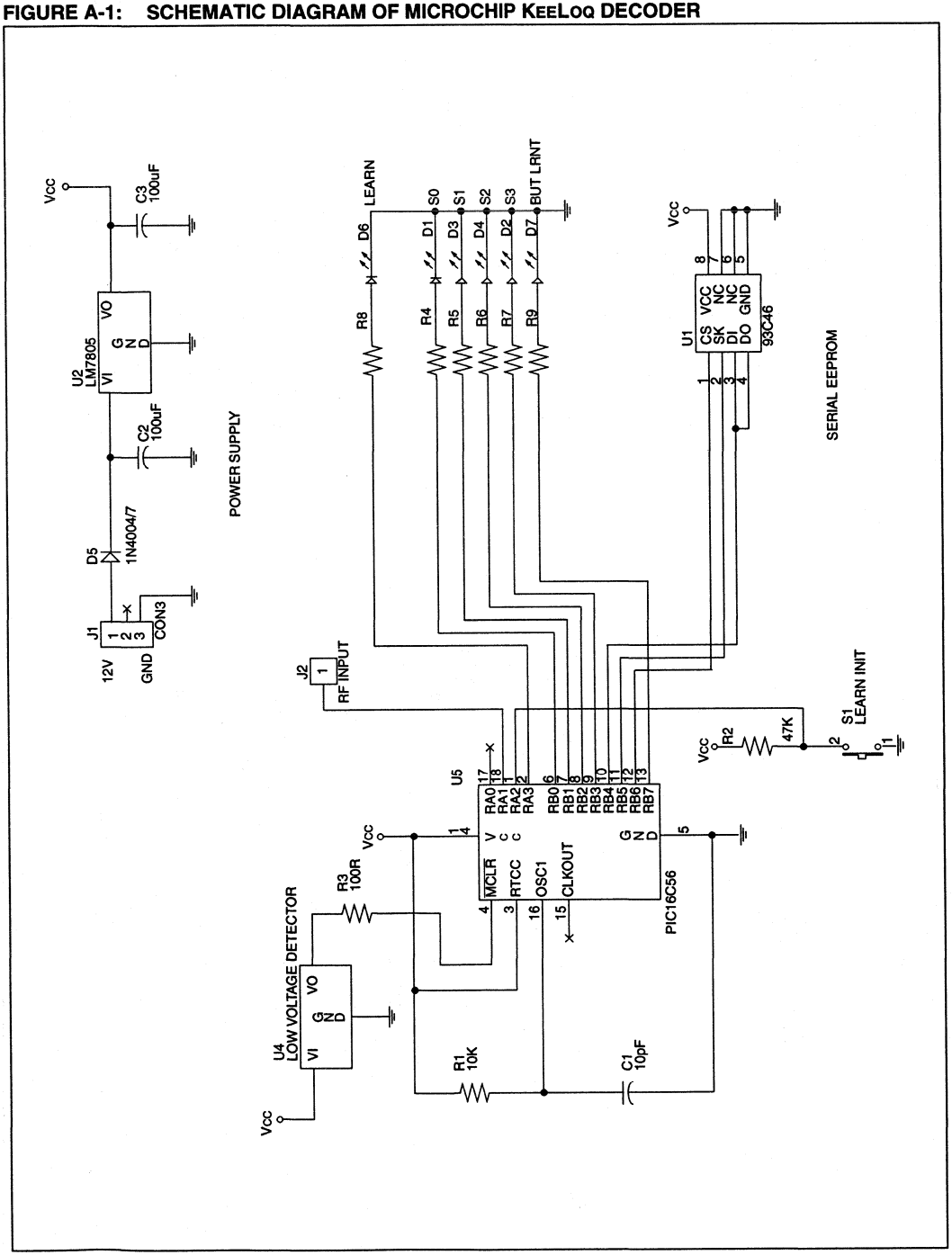
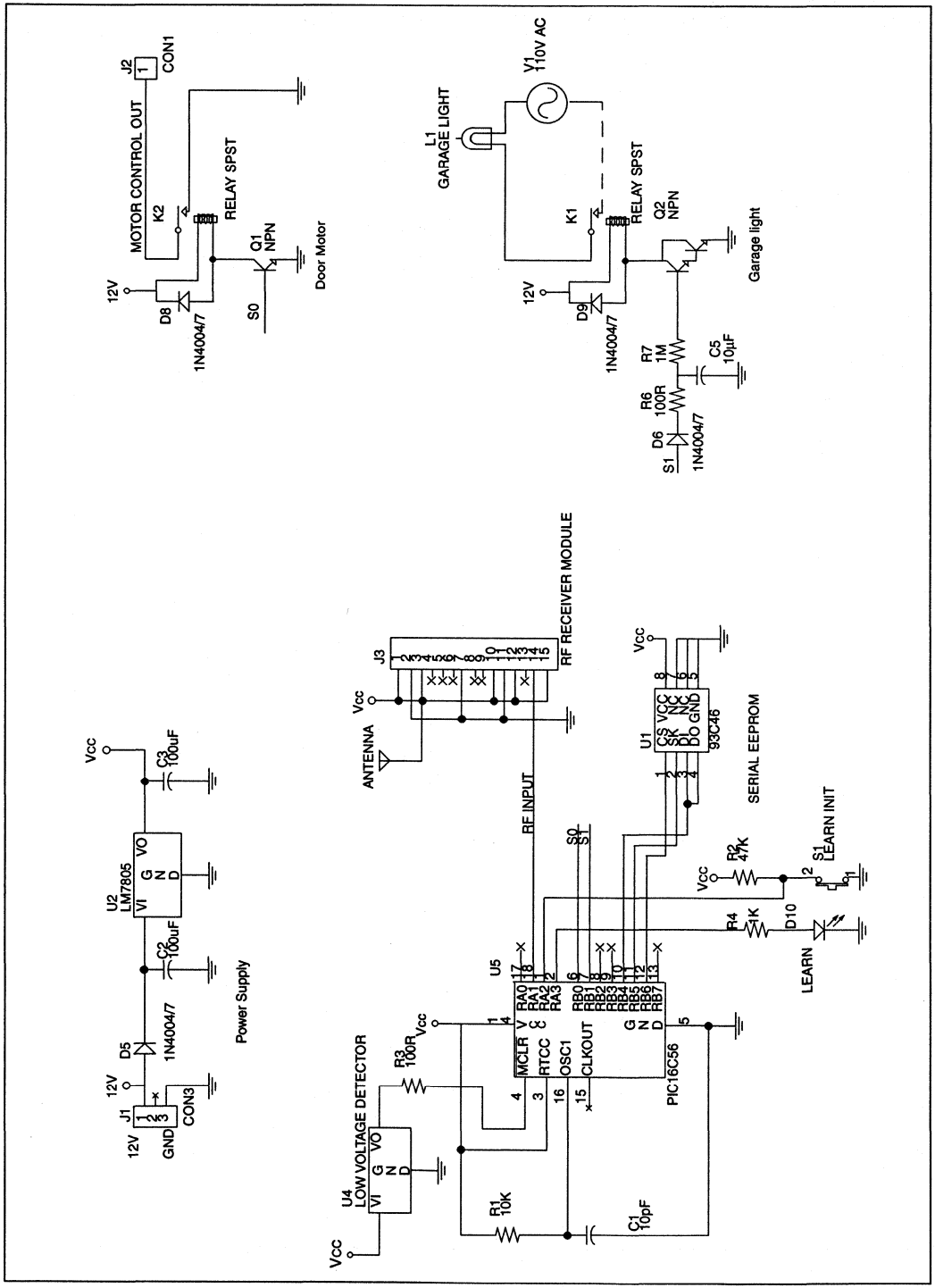


FIGURE A-2: TYPICAL GARAGE DOOR OPENER SCHEMATIC



© 1997 Microchip Technology Inc.

FIGURE A-3: HCS200/300/301 TRANSMITTER DESIGN

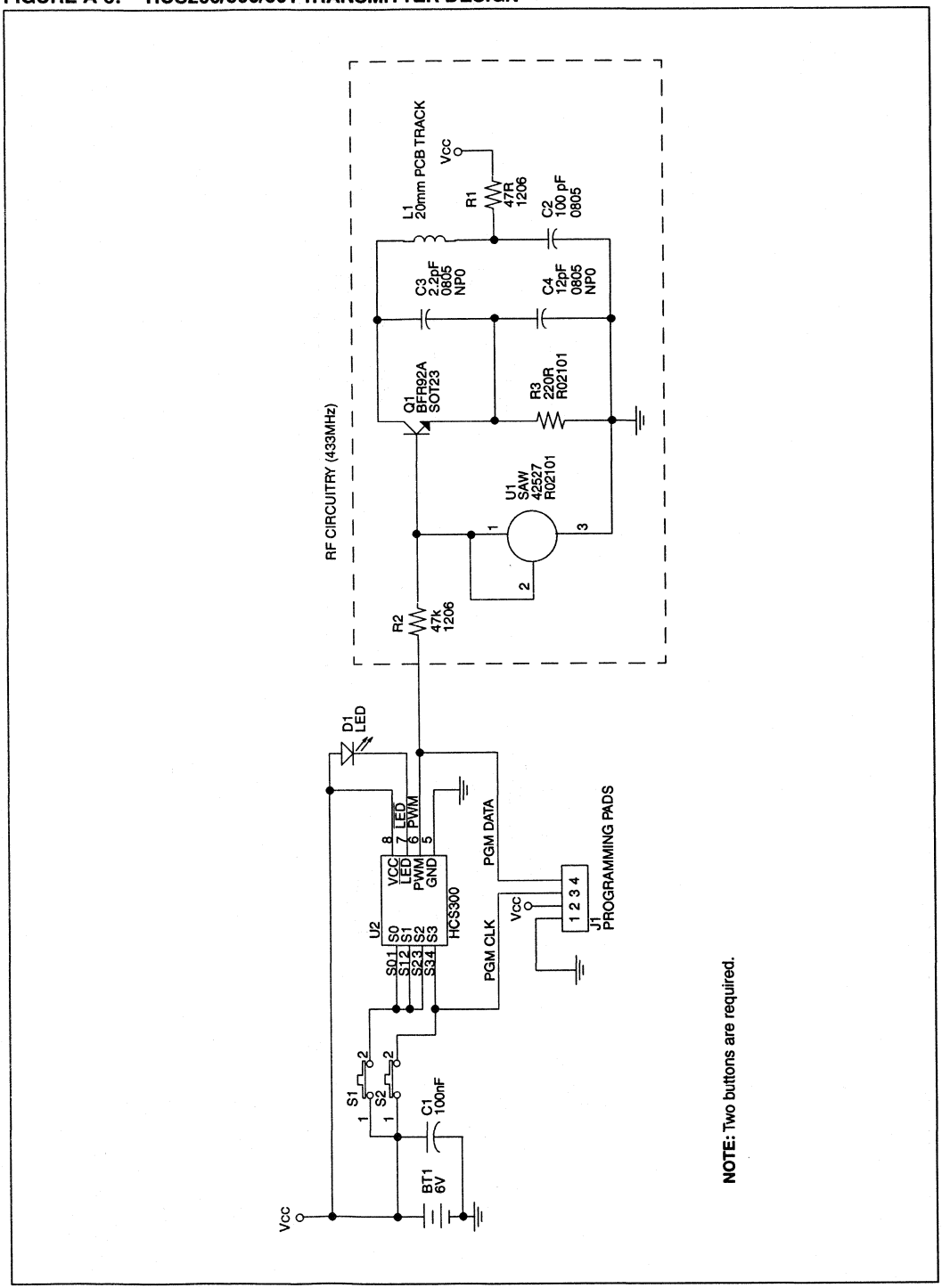
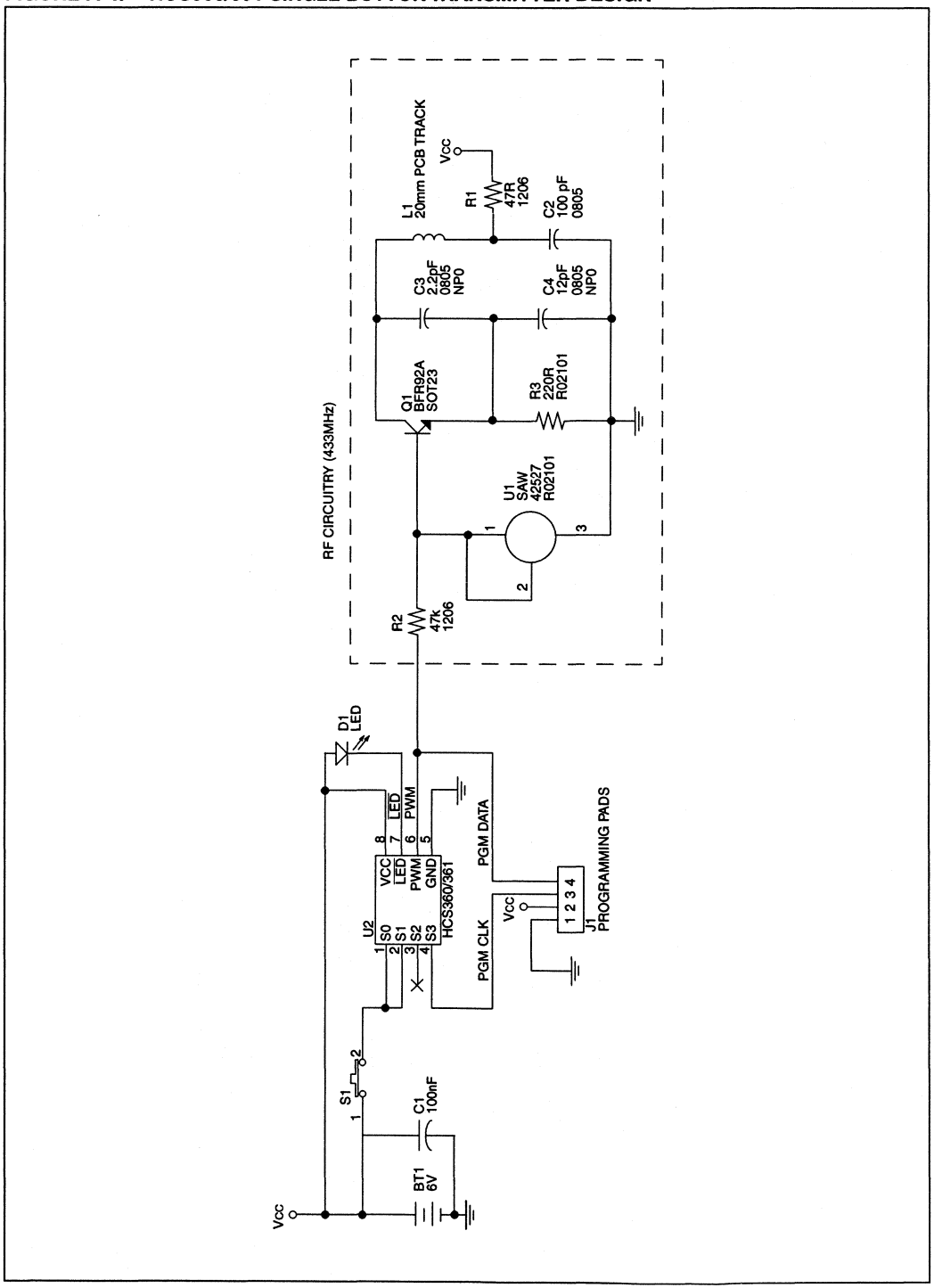


FIGURE A-4: HCS360/361 SINGLE BUTTON TRANSMITTER DESIGN



© 1997 Microchip Technology Inc.

MICROCHIP

AN663

KEELOQ® Simple Code Hopping Decoder

Author: Steven Dawson
Microchip Technology Inc.

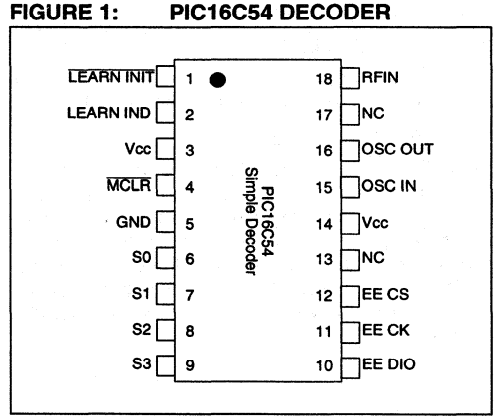
OVERVIEW

This application note fully describes the working of a code hopping decoder implemented on a Microchip PIC16C54 microcontroller. The PIC16C54 is smaller than the PIC16C56 used in the normal decoder (AN642) or the secure learn decoder (AN652). A simplified learning scheme with all encoders sharing a common key and a simplified counter synchronization scheme has been used to reduce the code space requirements. The use of a common key reduces the security of the system. The simple decoder can learn up to 15 encoders. This application note describes the various KEELOQ code hopping encoders that can be used with the decoder, the decoder hardware, and the various software modules comprising the system. The software can be used to implement a stand alone decoder or integrated with full function security systems. The decoder supports the Microchip HCS200, HCS300, HCS301, HCS360, and HCS361 KEELOQ Code Hopping Encoders.

KEY FEATURES

- PIC16C54
- Stand alone decoder
- Compatible with Microchip HCS200, HCS300, HCS301, HCS360, and HCS361 encoders
- Automatic bit rate detection
- Automatic encoder type detection
- Four function outputs
- Up to 15 learnable transmitters
- RC Oscillator
- Single press learn

FIGURE 1: PIC16C54 DECODER



6

Notice:

This is a non-restricted version of Application Note AN659 which is available under the KEELOQ License Agreement. The license agreement can be ordered from the Microchip Literature Center as DS40149.

KEELOQ is a registered trademark of Microchip Technology Inc.

© 1997 Microchip Technology Inc.

INTRODUCTION TO KEELOQ ENCODERS

All KEELOQ encoders use the KEELOQ code hopping technology to make each transmission by an encoder unique. The encoder transmissions have two parts. The first part changes each time the encoder is activated and is called the hopping code part and is encrypted. The second part is the unencrypted part of the transmission, principally containing the encoder's serial number identifying it to a decoder. Refer to DS91002, Introduction to KEELOQ.

Hopping Code

The hopping code contains function information, a discrimination value, and a synchronization counter. This information is encrypted by an encryption algorithm before being transmitted. A 64-bit encryption key is used by the encryption algorithm. If one bit in the data that is encrypted changes, the result is that an average of half the bits in the output will change. As a result, the hopping code changes dramatically for each transmission and can not be predicted.

Function Information

The encoder transmits up to four bits of function information. Up to 15 different functions are available.

Discrimination Value

Stored in the encoder EEPROM, this information can be used to check integrity of decryption operation by a decoder. If known information is inserted into the transmitted string before encryption, the same information can be used at the decoder to check whether the information has been decrypted correctly. 12 bits (including overflow bits) are available in the Microchip HCSXXX encoders.

Synchronization Counters

The transmitted word contains a 16-bit synchronization counter. The synchronization information is used at the decoder to determine whether a transmission is valid or is a repetition of a previous transmission. Previous codes are rejected to safeguard against code grabbers. The HSC300 and HCS301 encoders transmit two overflow bits which may be used to extend the range of the synchronization counter from 65,536 to 196,608 button operations. The HCS360 and HCS361 encoders transmit one overflow bit which can be used to extend the range of the synchronization counter from 65,536 to 131,071 button operations.

Unencrypted Code

Serial Number

The encoder's serial number is transmitted every time the button is pressed. The serial number is transmitted unencrypted as part of the transmission and serves to identify the encoder to the decoder.

Other Status and Function Information

The HCS200, HCS300, and HCS301 encoders include provision for four bits of function information and two status bits in the fixed code portion of its transmission. The two status bits indicate whether a repeated transmission is being sent, and whether the battery voltage is low. The HCS200 does not send repeated transmission information, and the bit is permanently set to '0'.

The HCS360/361 encoders transmit two bits that are used as a Cyclic Redundancy check. These bits can be used to check the integrity of the reception. Additionally, the HCS360 and HCS361 encoders can extend the length of the serial number from 28 bits to 32 bits, replacing the unencrypted function code.

Transmission Format Summary

Table 1 contains a summary of the information contained in transmissions from each of the KEELOQ encoders that can be learned by the Microchip decoder.

© 1997 Microchip Technology Inc.

FIGURE 2: DECODER BLOCK DIAGRAM

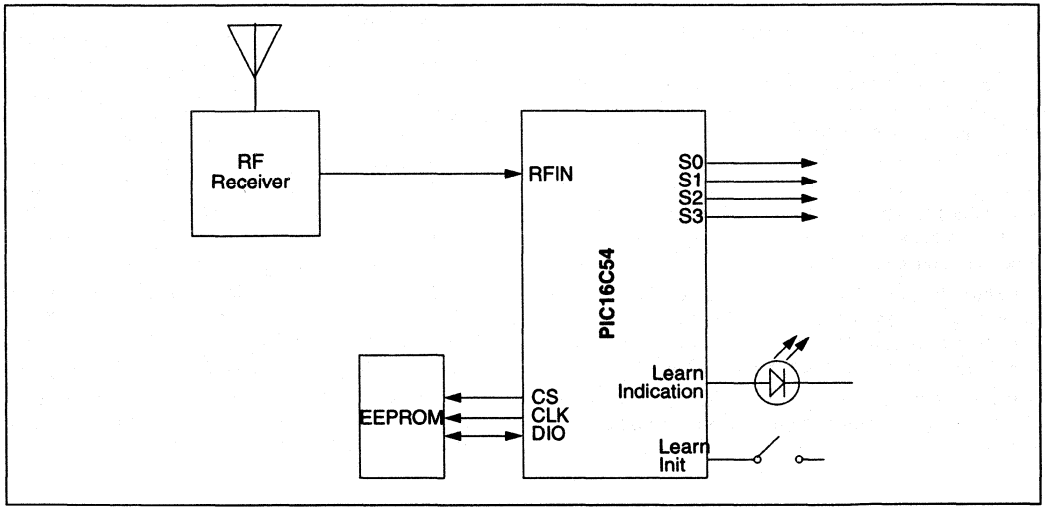


TABLE 1: KEELOQ ENCODER TRANSMISSION SUMMARY

	HCS200[1]	HCS300/301	HCS360/361
Total transmission length	66 bits	66 bits	67 bits
Hopping code portion (total length)	32 bits	32 bits	32 bits
Function bits	3 bits (+1[1])	4 bits	4 bits
Discrimination bits	12 bits	12 bits	12 bits
Synchronization	16 bits	16 bits	16 bits
Fixed portion (total length)	34 bits	34 bits	35 bits
Serial number	28 bits	28 bits	28/32 bits[2]
Function bits	3 bits (+1[1])	4 bits	4/0 bits[2]
Status bits			
VLOW	1 bit	1 bit	1 bit
RPT	0	1 bit	0
CRC	0	0	2 bits

[1]The HCS200 transmissions are padded to retain compatibility with the HCS300/301 encoders.
[2]User selectable with a 32-bit serial number or a 28-bit serial number, and 4 unencrypted function bits.

The code hopping portion is always transmitted first (LSB to MSB), followed by the fixed portion (LSB to MSB).

TABLE 2: HCS200† AND HCS300/301 TRANSMISSION FORMAT

Hop Code			Serial Number			Function	Status
LSB	32 bits	MSB	LSB	28 bits	MSB	4 bits	2 bits

†The HCS200 transmissions are padded to retain compatibility with the HCS300/301 encoders.

TABLE 3: HCS360/361 TRANSMISSION FORMAT

Hop Code			Serial Number			Function	Status
LSB	32 bits	MSB	LSB	28/32 bits	MSB	4/0 bits	3 bits

Bits are transmitted from left to right (i.e., the code hopping part first).

© 1997 Microchip Technology Inc.

PWM Format

In general, all KEELOQ encoders share a common transmission format:

A **preamble** to improve biasing of decision thresholds in super-regenerative receivers. The preamble consists of alternate on and off periods, each lasting as long as a single elemental period.

A **calibration** header consisting of a low period of 10 elemental periods. Calibration actions should be performed on the low period of the header to ensure correct operation with header chopping.

A **string** of 66 or 67 pulse-width modulated bits, each consisting of three elements. The first element is high, the second contains the data transmitted and is either high or low, the third element is always low.

A **guard** period is usually left between the transmissions. During this period nothing is transmitted by the encoder.

Figure 3 shows the sampling points when sampling the data bits. The first and last elements are used exclusively to verify the integrity of the received symbol. The first element (sample point A) is always high, the second (sample point B) is the complement of the data bit being sent, and the final element (sample point C) is always low. Because the period between the low portion of a bit (sample point C) and the rising edge of the following bit (sample point X) can vary, the rising edge of the first element (sample point X) is used to resynchronize the receiving routine to each incoming bit.

If random noise is being received, the probability of a set of three samples producing a valid combination is only $2^{-2} = 1/4$. For a string of 66 bits, the corresponding figure is $2^{-132} < 2 \times 10^{-40}$.

Integrity checking on incoming signals is important. Code hopping signals require significant processing, as well as EEPROM access, to decrypt. Unnecessary processing can be avoided by not attempting to decrypt incoming codes that have bit errors.

FIGURE 3: KEELOQ PWM TRANSMISSION FORMAT

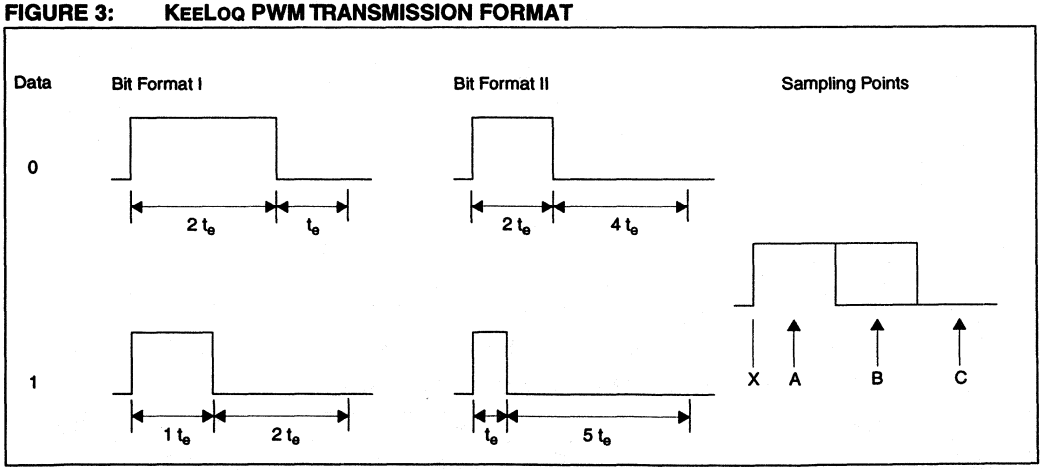


TABLE 4: TRANSMISSION FORMATS

Encoder	Preamble	Header Chopping	PWM Bit Format		Hopping Code Length	Unencrypted Code Length
			I	II		
HCS200	•	•	•	—	32 bits	34 bits
HCS300/301	•	•	•	—	32 bits	34 bits
HCS360	•	•	•	—	32 bits	35 bits
HCS361	•	•	•	•	32 bits	35 bits

© 1997 Microchip Technology Inc.

DECODER IMPLEMENTATION

The decoder is implemented on a PIC16C54 RISC microcontroller and a 93C46 EEPROM as shown in the decoder schematic in Figure 9. However, this solution can be implemented in any PICmicro™ microcontroller with at least 500 words of program memory. The operating frequency of the controller is 4 MHz. The microcontroller is used to capture transmissions from the various encoders, decrypt transmissions captured, and check the validity of the transmission based on the information in the decrypted transmission and information stored in the EEPROM. If a transmission from a valid encoder is received, the decoder activates the outputs dictated by the transmission.

Encoder information, such as serial number and synchronization information is stored externally in an EEPROM. The EEPROM used is a Microchip 93C46 Microwire® Serial EEPROM. The information stored in the EEPROM is encrypted to protect the contents. The EEPROM encryption is less secure than the KEELOQ code hopping algorithm.

As can be seen from the section on encoder transmissions there are differences in the transmission formats of the different encoders that can be used with the decoder. The following section summarizes how the differences in transmitted data are dealt with by the decoder.

As the serial number information follows after the code hopping portion of the transmission, any number of serial number bits can be received and processed. In the Microchip decoder described, the complete serial number (28 bits) is stored.

After matching the received and stored serial number, validation of a received transmission consists of two steps. The first includes checking the integrity of the decryption operation. Here the decoder compares the least significant 8-bits of the discrimination value received with the least significant 8-bits of the serial number. The discrimination value stored with the HCS300/301/360/361 includes overflow bits and user bits.

The second portion of validation involves checking synchronization information for that particular encoder. The synchronization counter transmitted by all encoders is 16 bits long. Two copies of the full synchronization counter are stored for all valid encoders. The storing of two copies of the synchronization information protects the decoder from loosing synchronization with an encoder if one of the counters is corrupted.

6

TABLE 5: MICROCHIP DECODER FUNCTIONAL INPUTS AND OUTPUTS

Mnemonic	Pin Number	Input / Output	Function
RF IN	18	I	Demodulated PWM signal from RF receiver. The decoder uses this input to receive encoder transmissions.
$\overline{\text{LEARN INIT}}$	1	I	Input to initiate learning, active low.
LEARN IND	2	O	Output to show the status of the learn process (in an integrated system this will be combined with the system status indicator).
S0, S1, S2, S3	6, 7, 8, 9	O	Function outputs—corresponds to encoder input pins.

Microwire is registered trademark of Motorola.

© 1997 Microchip Technology Inc.

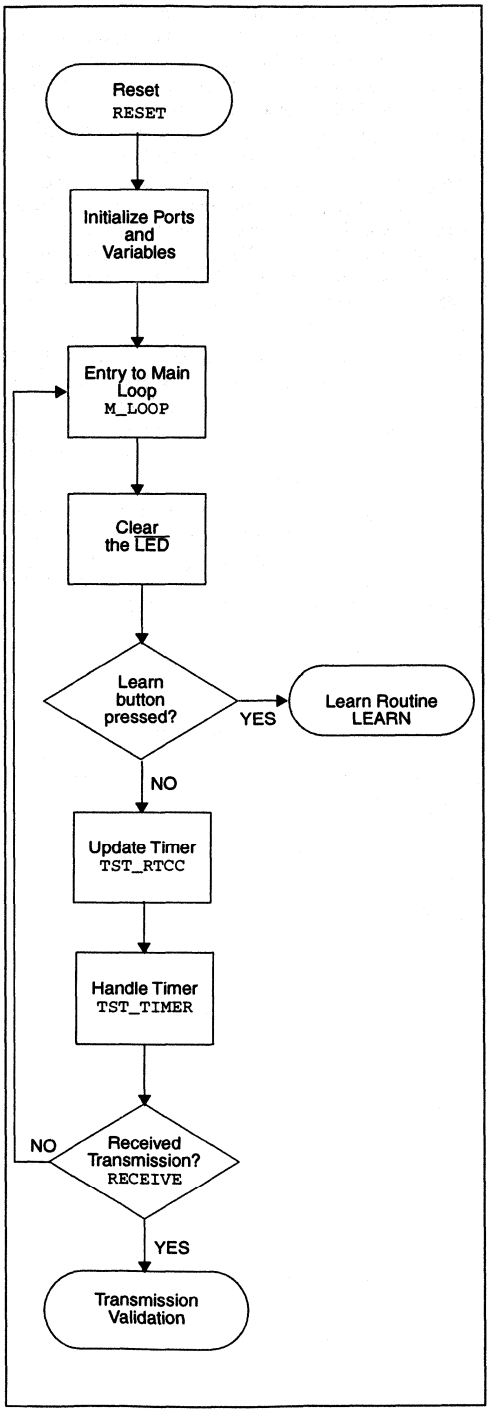
PROGRAM FLOW

The software for the Microchip decoder has been written for the PIC16C54 microcontroller. The compiler used is MPASM version 01.30.01. The operating frequency of the PIC16C54 is 4 MHz. The clock speed should be kept as close as possible to 4 MHz as the reception routine (RECEIVE) is dependent on the 4 MHz clock for correct functioning. Other decoder functions that rely on a 4 MHz clock speed are the hold times of the various outputs and time-outs. The main program flow is described here. Detailed descriptions of individual functions can be found further in the application note.

After startup, the encoder enters the main loop where it spends most of its time. The main loop checks to see if the learn button is being activated. If so, the decoder enters the learn mode described in the "Learn" in AN659.

If learn has not been initiated, the microcontroller checks for transmissions from encoders (RECEIVE) as described on page 55. If a transmission from an encoder has successfully been received, the microcontroller validates the transmission received as described in the "Transmission Validation" section in AN659. If the transmission received is a valid transmission from an encoder learned into the system, the system sets the appropriate outputs (M_OUTPUT).

FIGURE 4: MICROCHIP DECODER MAIN PROGRAM FLOW



Microwire is a registered trademark of Motorola.

© 1997 Microchip Technology Inc.

FUNCTIONAL MODULES

Reception

The reception routine (RECEIVE) is based on a reliable algorithm which has successfully been used in previous implementations of KEELOQ decoders. Automatic bit rate detection is used to compensate for variations in bit rate of different encoders of a specific type, as well as the differences in bit rate between different encoders (HCS200, HCS300, and HCS360). The reception routine is able to receive 64-bit transmissions. The decoder only uses the first 60 bits received. The remaining bits are ignored.

The reception algorithm performs a number of functions when an output is detected from the receiver. Figure 5 gives all the major sampling points in the reception algorithm.

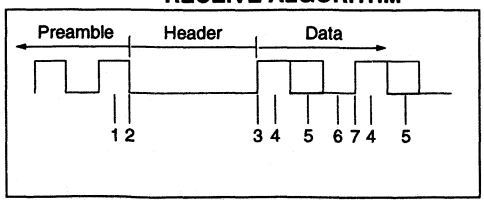
The reception algorithm calibrates on the low period of the header to determine the actual elemental period for the transmission being received. The required elemental period is 10% of the low header period. In Figure 5 the header calibration sample points are marked 1 through 3. The calibration flow chart (Figure 6) shows at what points in the program samples 1, 2, and 3 are taken.

Elemental periods outside the capture range of the algorithm (either too long or too short) are rejected, since they are due either to noise or to reception of an incomplete signal.

Using the determined elemental period, three samples after the first rising edge (sample 3) following the header are taken. The first sample is taken half an elemental period after the rising edge (Sample 4); the second, one elemental period later (Sample 5), and the third, another one elemental period later (Sample 6). The first sample must be high, the second could be either high or low, and the third sample must be low. If either the first or the third sample is not as expected, the attempt at capturing a transmission is abandoned. In Figure 5, the data sample points are points 4 through 6. The flow chart describing data reception (Figure 7) shows where in the code the samples are taken.

If all 64 bits have been captured, each with the correct first and third elements, the transmission can be assumed to be correct, and decryption can commence. The receiving routine should be called often enough to ensure that the high portion in the header is not missed (Sample 1, Figure 5).

FIGURE 5: SAMPLING POINTS USED IN RECEIVE ALGORITHM



In systems where the reception routine is called to check if there is activity on the receiver input, the routine should poll the input for a valid transmission for at least the time taken to complete one transmission if activity is detected on the input line. This makes provision for the reception routine being called while a transmission is in progress. Having missed the first header, the first transmission will be invalid and be discarded. The decoder should continue sampling the input through the guard time in order to catch the next header and transmission (i.e., for a decoder designed to capture HCS300 transmissions the time spent polling for a valid transmission should be at least 100 ms if activity is detected in the input line).

Reception Algorithm Flow Chart

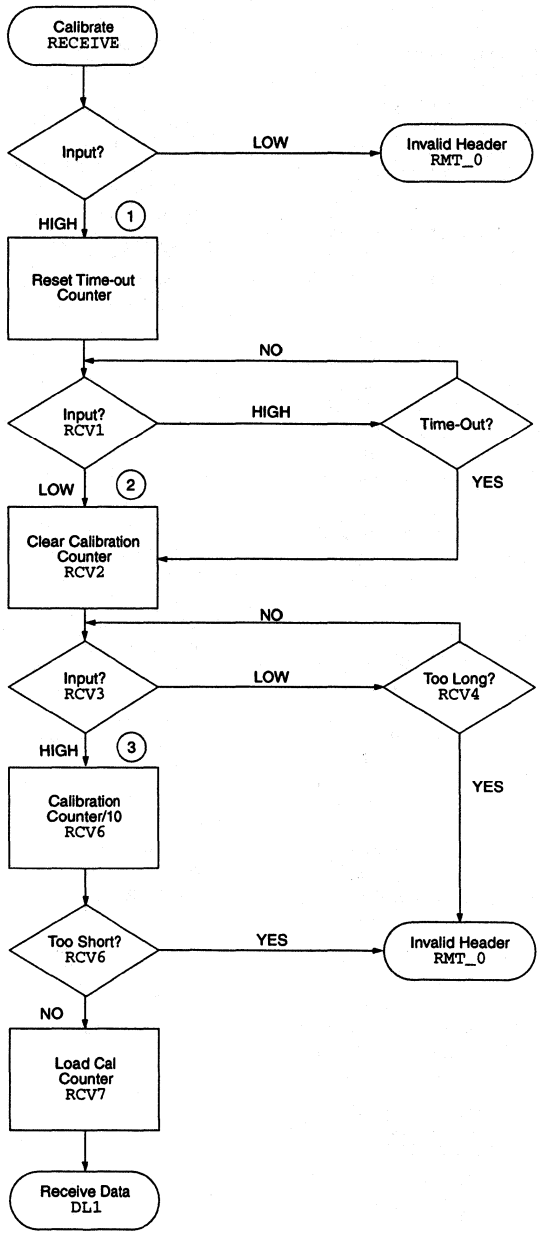
The flow chart in Figure 6 describes the calibration routine which is used to determine the actual transmission rate of the encoder so that the decoder can compensate for deviations from nominal timing. There are four different exit points, each of which should branch to a point in the program where housekeeping and input monitoring can be resumed. There is only one exit point for a valid calibration operation (RCV7). At this point, it is assumed that a valid header has been received and that a string of data bits will follow.

The second flow chart (Figure 7) handles the reception of bits once the calibration routine has been successfully completed. The data bits are all sampled three times each to ensure that a noise free transmission has been received. The reception routine uses the calibrated elemental period, determined in the calibration routine, to ensure that the samples are correctly spaced. The routine resynchronizes itself on the rising flank of each bit. Only 60 bits of the data received are used by the decoder described, the decoder ignores the unencrypted function code and the status bits.

If the control samples in a given bit are sampled correctly (i.e., the first element is high and the last element is low), the routine checks whether 66 bits have been received correctly. If not, the routine returns to the calling procedure.

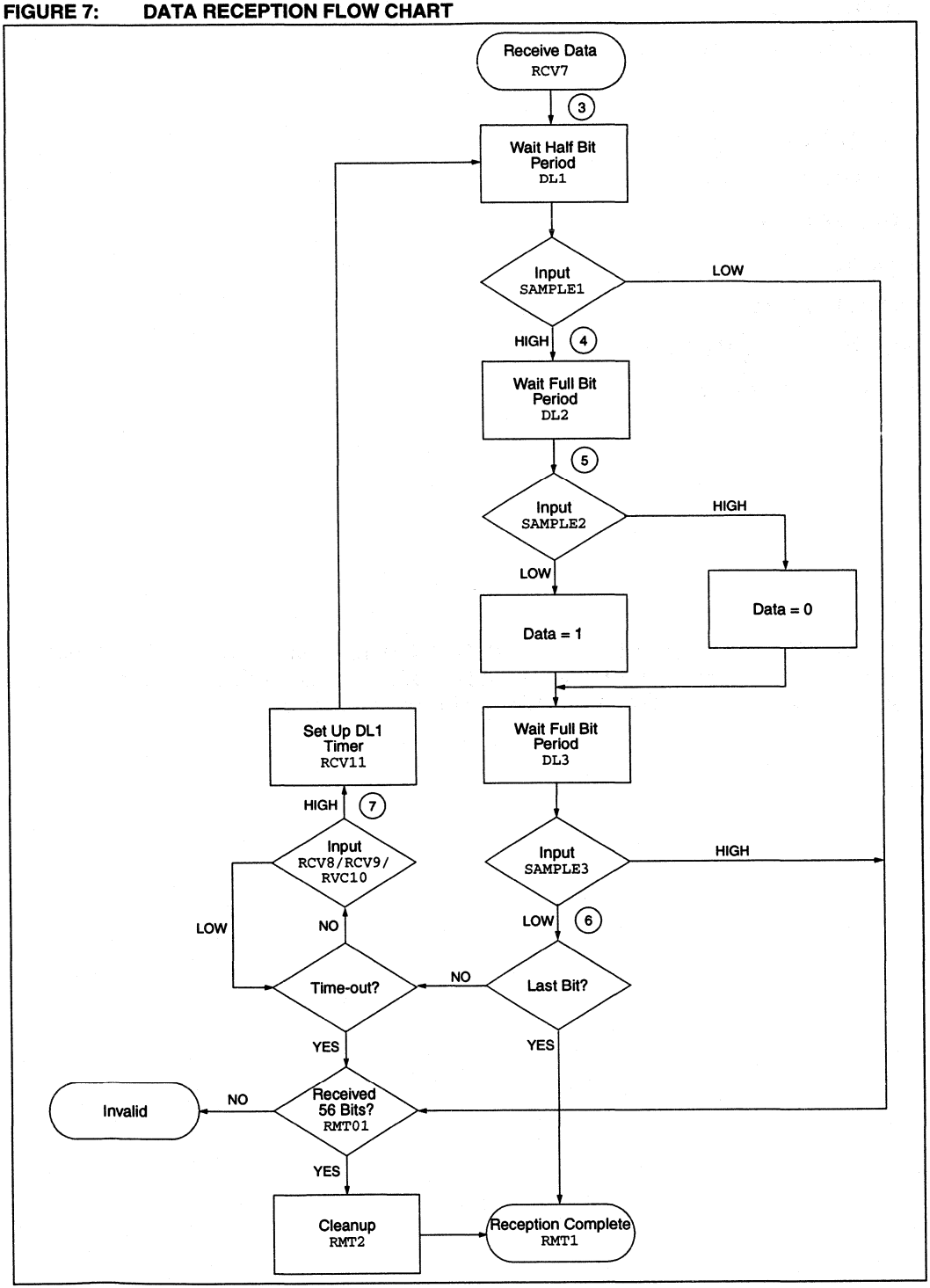
© 1997 Microchip Technology Inc.

FIGURE 6: CALIBRATION FLOW CHART



© 1997 Microchip Technology Inc.

FIGURE 7: DATA RECEPTION FLOW CHART



VALIDATION

SYNCHRONIZATION

FUNCTION INTERPRETATION

OUTPUT ACTIVATION

KEY GENERATION

DECRYPTION

LEARN

ROM MEMORY MAP

EEPROM MEMORY MAP

RAM MEMORY MAP

The confidential and proprietary information contained in this section of AN659 has been removed. The full application note is available under a license agreement and can be ordered as DS40149 from Microchip Technology Inc.

© 1997 Microchip Technology Inc.

DEVICE PINOUTS

The device used in the application note is a PIC16C54 PDIP or SOIC.

TABLE 6: DEVICE PINOUTS

PIN	PIC16C54 Function	Decoder Function	PIN	PIC16C54 Function	Decoder Function
1	Port A Bit 2	LEARN Input	18	Port A Bit 1	RF Input
2	Port A Bit 3	LEARN Indicator	17	Port A Bit 0	Not used
3	TIMER0	Connect to VDD	16	Osc In	RC osc (4 MHz)
4	$\overline{MCLR}$	Brown out detect	15	Osc Out	
5	GND	Ground	14	VDD	+5V supply
6	Port B Bit 0	S0	13	Port B Bit 7	Not Used
7	Port B Bit 1	S1	12	Port B Bit 6	CS (93C46, pin 1))
8	Port B Bit 2	S2	11	Port B Bit 5	CLK (93C46, pin 2)
9	Port B Bit 3	S3	10	Port B Bit 4	DIO (93C46, pin 3 & 4)

TIMING PARAMETERS

TABLE 7: TIMING PARAMETERS

Parameter	Typical	Unit
Output activation duration	524	ms
Output pause if new function code received	131	ms
Erase all duration	8.4	s
Learn mode time-out	33.6	s
Learn failure $\overline{LED}$ on duration	1	s

6

SOURCE CODE LISTING

A diskette is supplied containing source code for the Microchip decoder in the file SIMDEC**.ASM. The code has been compiled using MPASM v01.30.01. Certain functions are dependent on the oscillator speed for correct functioning. Examples of time dependent functions include RECEIVE and TST_TIMER. The PIC16C54 Microcontroller should run at 4 MHz.

TABLE 8: LIST OF IMPORTANT FUNCTIONS

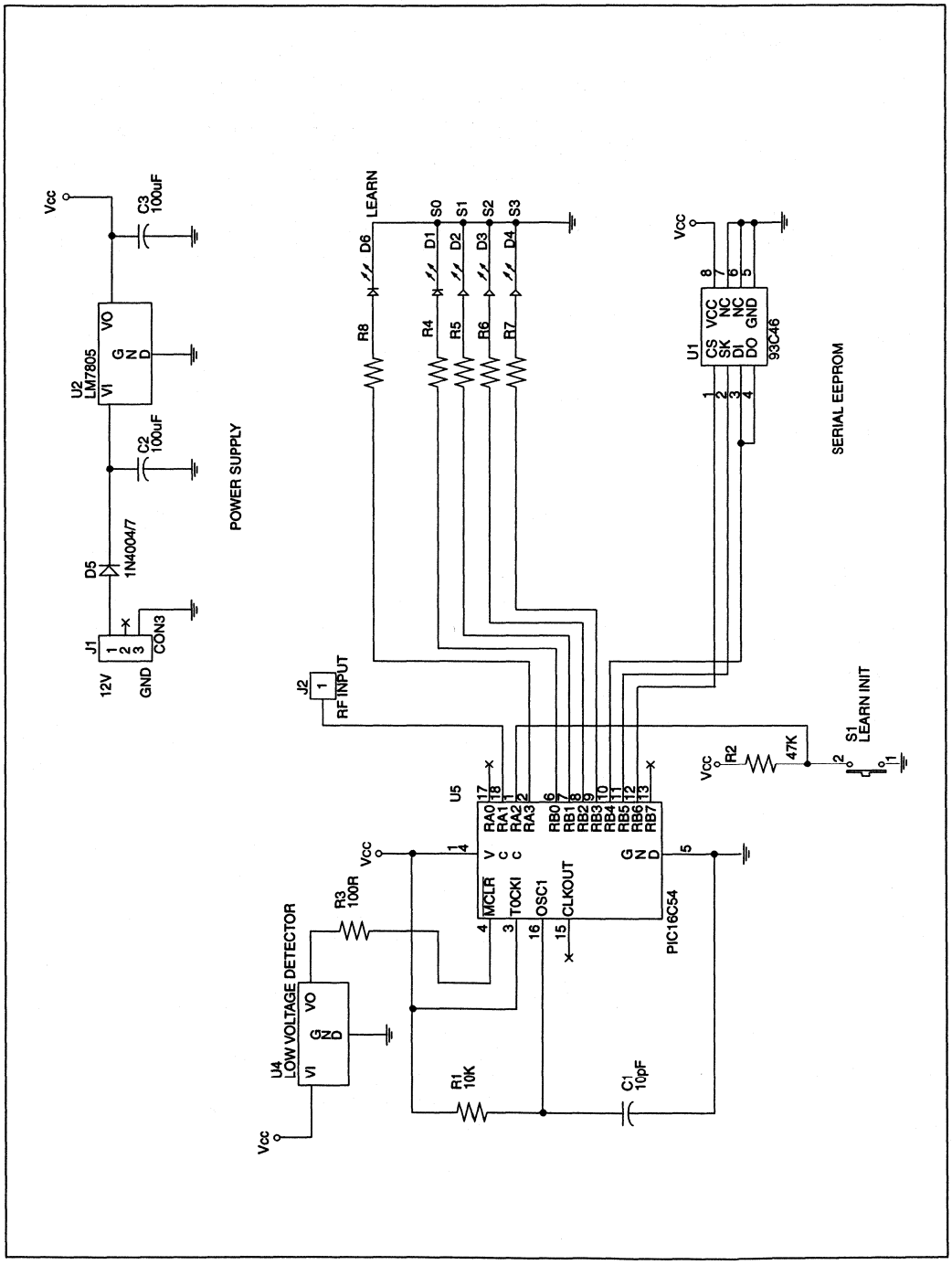
Function Name	Description
DECRYPT	Decryption routine for Hop Code.
EEREAD	The data in the EEPROM at ADDRESS is read to TMP1 and TMP2 (Note).
EEWRITE	The data in TMP1, and TMP2 is written to the EEPROM at ADDRESS (Note).
M_DIS	Check discrimination value.
M_CNT	Check synchronization (counter) values.
RECEIVE	Start of the RF reception routine.
LEARN	Learn mode.
TST_RTCC	Check Timer0 and update CNT_LW and CNT_HI.
TST_TIMER	Check CNT_LW and CNT_HI and do whatever real-time tasks that are required.

Note: TMP1, TMP2 and ADDRESS are user defined registers.

© 1997 Microchip Technology Inc.

APPENDIX A: SCHEMATIC DIAGRAMS

FIGURE A-1: SCHEMATIC DIAGRAM OF MICROCHIP KEELOQ DECODER



© 1997 Microchip Technology Inc.

FIGURE A-2: TYPICAL GARAGE DOOR OPENER SCHEMATIC

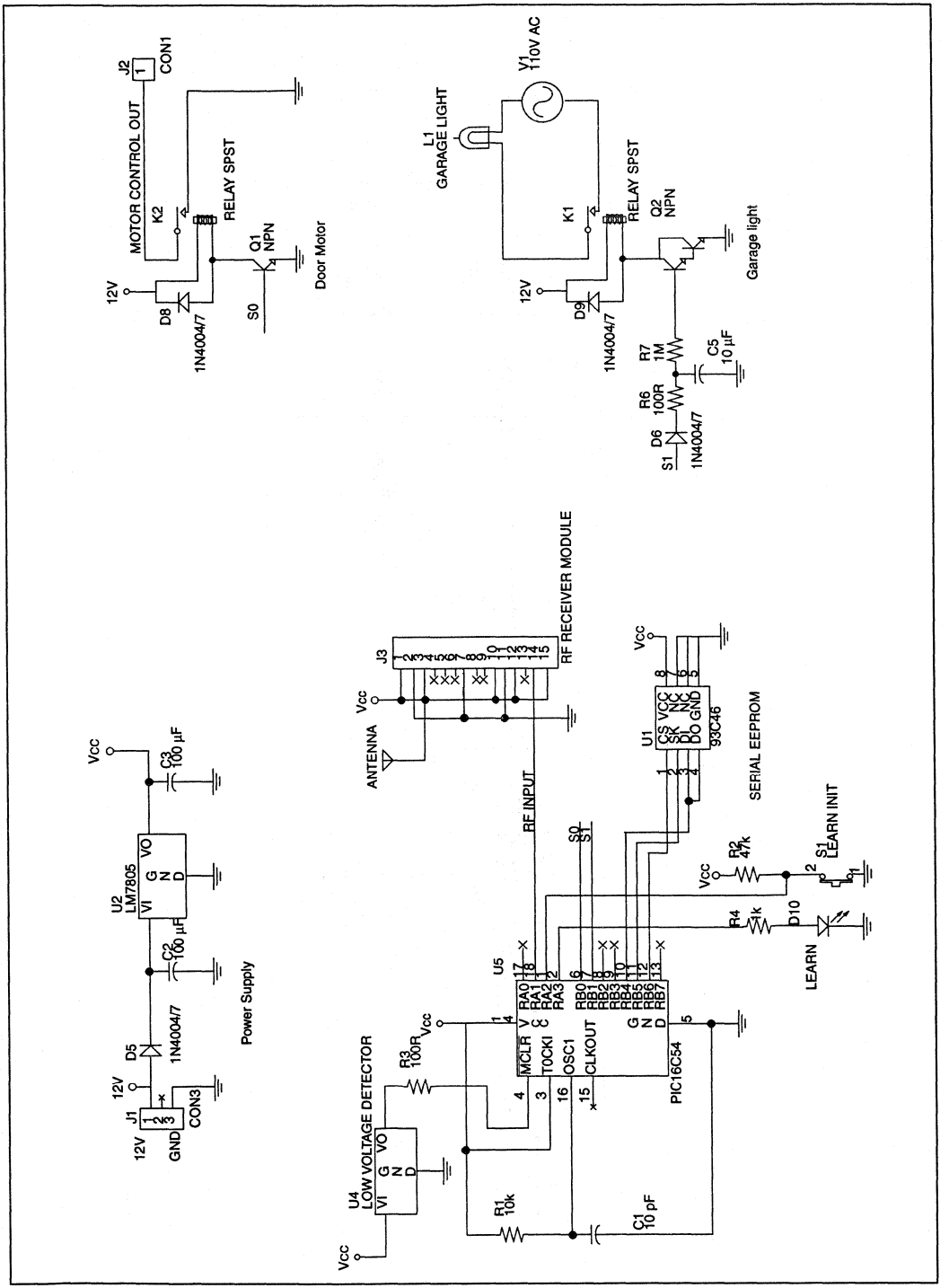
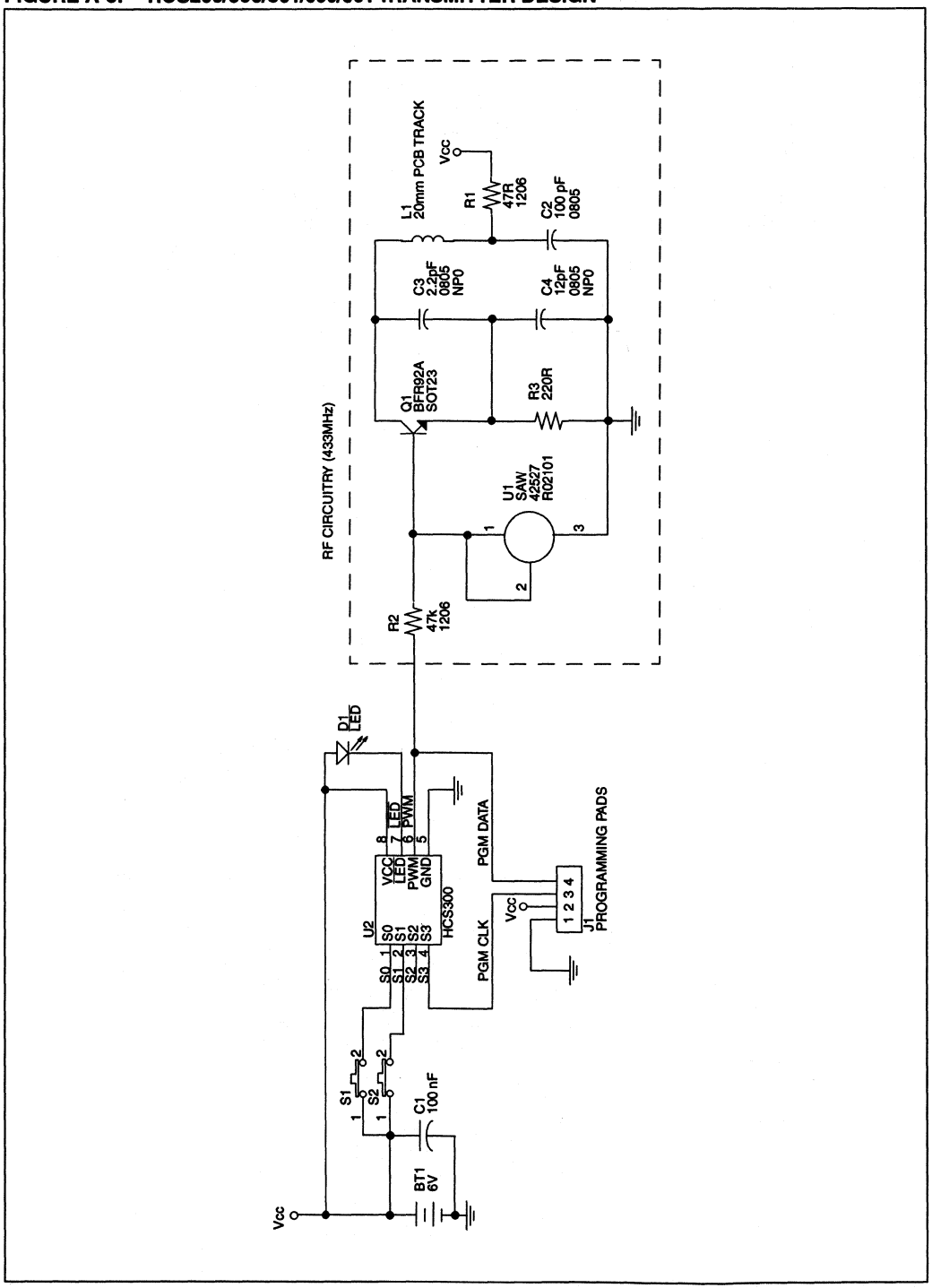


FIGURE A-3: HCS200/300/301/360/361 TRANSMITTER DESIGN



© 1997 Microchip Technology Inc.

MICROCHIP

AN665

Using KEELOQ® to Generate Hopping Passwords

Author: *Lucio Di Jasio*
Arizona Microchip Technology, Italy

INTRODUCTION

The purpose of this application note is to demonstrate how KEELOQ code hopping technology can be conveniently employed to implement an automatic code hopping password generator/keypad. Using a PIC12C508, the hopping code produced by an HCS300 is converted to a string of 16 hex digits. This string is then transferred to the PC via the keyboard line, thereby emulating the actual pressure of a sequence of keys on a standard PC/AT® keyboard. Since this conversion process is transparent to any application, it appears as if the user is simply typing on a PC/AT-type keyboard.

An ideal situation for implementing this application would be in creating a "super password" for general, access-control secure logins when transmitting information onto the internet (i.e., through a browser) or a Java applet.

THE "HOPPING" ADVANTAGE

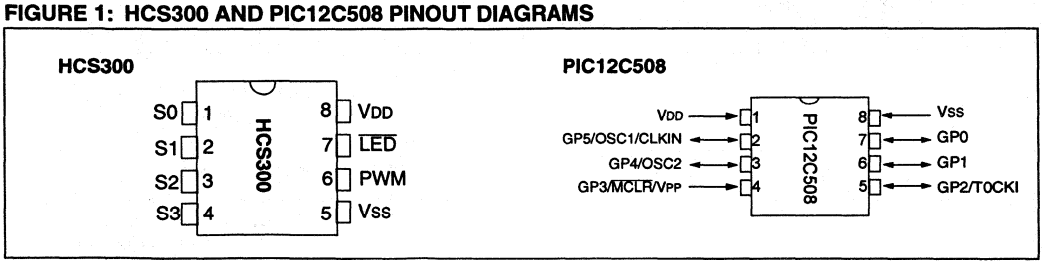
Password-based access control systems are very popular today, but the level of security they provide are often overestimated. Being basically a unidirectional transmission, a password-based system has two very important shortcomings which can lead to unauthorized access: the code is fixed, and the number of possible combinations is relatively low.

The growing speed of communication lines and the computing power of available systems increases the chance of a brute force attack or "code scanning." The use of unsecure means of transmission, where code "grabbing" is possible (i.e., a typical modem connection over phone lines), can make the use of a fixed code highly undesirable. Note that these are the same situation that led to the introduction of the "code hopping" concept in the remote control market.

The basic idea is to have the access code change each time it is used through a sequence where the new codes cannot be predicted even knowing a very large number of previously used ones. Producing such a sequence requires the use of a solid encryption engine. Microchip Technology is currently offering a broad range of encoders based on the proprietary KEELOQ code hopping technology. These encoders make producing a code hopping remote control easy, but as we will see, can also be conveniently used to add the hopping advantage to old password based access control systems in a transparent way.

6

FIGURE 1: HCS300 AND PIC12C508 PINOUT DIAGRAMS



KEELOQ is a registered trademark of Microchip Technology Inc.
PC/AT is a registered trademark of International Business Machines Corp.

© 1997 Microchip Technology Inc.

INTRODUCTION TO KEELOQ ENCODERS

All KEELOQ encoders use the KEELOQ code hopping technology to make each transmission by an encoder unique. The encoder transmissions have two parts. The first part changes each time the encoder is activated and is called the code hopping part and is encrypted. The second part is the unencrypted part of the transmission, principally containing the encoder serial number identifying it to a decoder.

The code hopping contains function information, a discrimination value, and a synchronization counter. This information is encrypted by an encryption algorithm before being transmitted. A 64-bit encryption key is used by the encryption algorithm. If one bit in the data that is encrypted changes, the result is that an average of half the bits in the output will change. As a result, the code hopping changes dramatically for each transmission and can not be predicted.

The synchronization information is used at the decoder to determine whether a transmission is valid or is a repetition of a previous transmission. Previous codes are rejected to safeguard against code grabbers.

The HSC300 and HCS301 encoders transmit two overflow bits which may be used to extend the range of the synchronization counter from 65,536 to 196,608 button operations. The HCS300 and HCS301 encoders include provision for four bits of function information and two status bits in the fixed code portion of its transmission. The two status bits indicate whether a repeated transmission is being sent, and whether the battery voltage is low.

The Microchip HCSXXX encoders all have the ability to transmit a fixed seed. The seed value is programmed into the encoder when the encoder is first initialized along with the counters, key, serial number, and other information. The seed length differs from encoder to encoder, with the HCS300 and HCS301 having a 32-bit seed.

FIGURE 2: KEELOQ ENCODER CODE WORD TRANSMISSION FORMAT

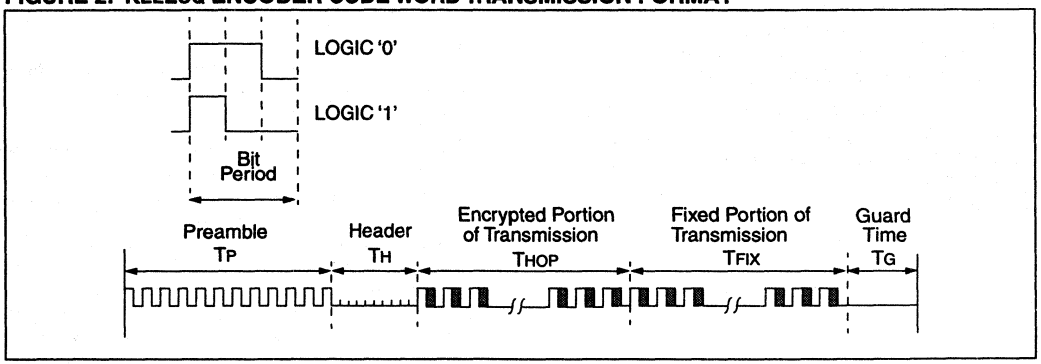
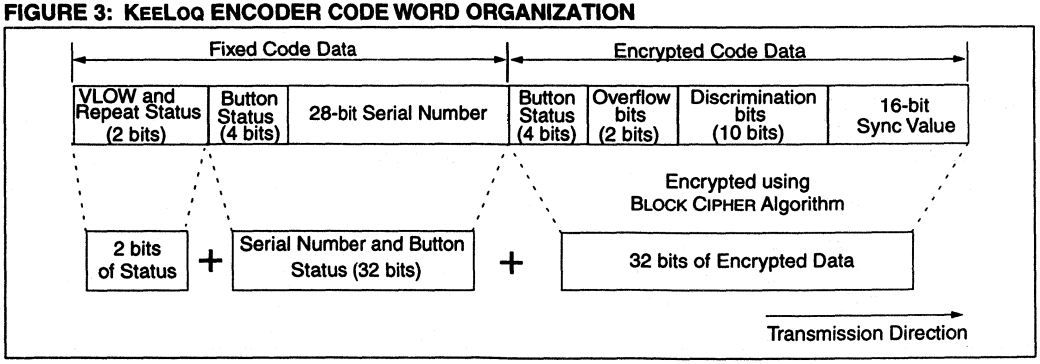
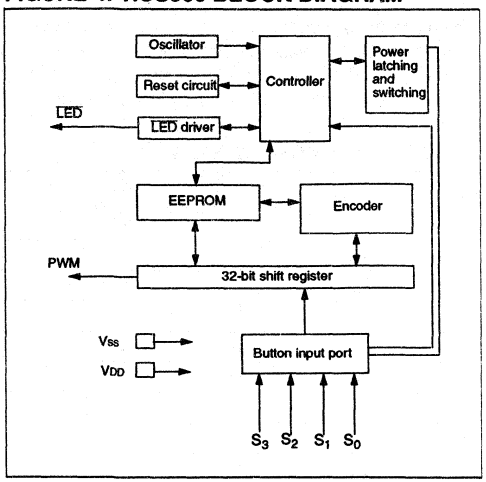


FIGURE 3: KEELOQ ENCODER CODE WORD ORGANIZATION



© 1997 Microchip Technology Inc.

FIGURE 4: HCS300 BLOCK DIAGRAM

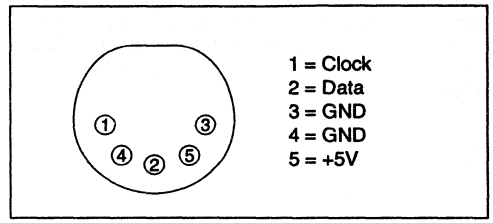


The IBM PC-AT® Keyboard Protocol

IBM® was the first to introduce the synchronous serial protocol most of today's PC-ATs use to communicate with a keyboard. This now-standard, 5-pole shielding connector (Figure 5) carries the clock line, data line, ground, and +5V power supply in order to transmit data bidirectionally from the keyboard to the PC.

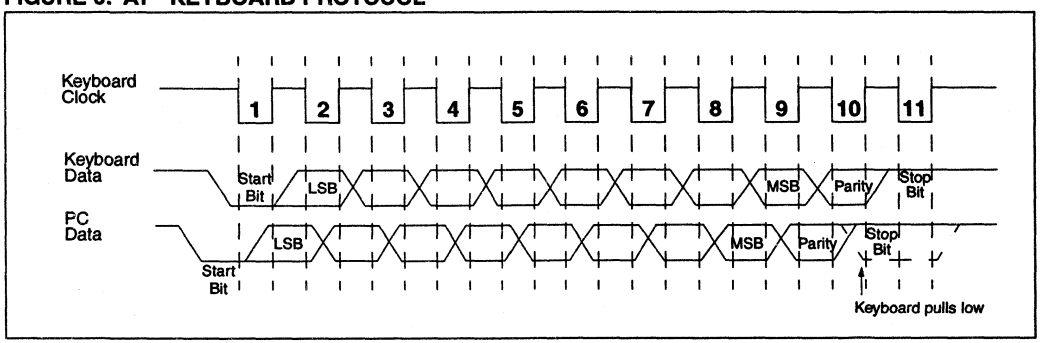
Typically, data travelling from the keyboard to the PC is accomplished by either key pressure or release information. However, some configuration data (i.e., repeat, delay, and rate) can flow in the opposite direction – for example, during a system boot. The keyboard drives the clock line by using open collector drivers. To disable the keyboard, the PC can keep the clock line low. If the data line is held low by the PC while the clock line is high, the computer transmits a requests to send, and the keyboard goes into receive mode. The keyboard is only allowed to send data when both the clock line and data line are high.

FIGURE 5: STANDARD 5-POLE CONNECTOR



6

FIGURE 6: AT® KEYBOARD PROTOCOL



IBM PC-AT, IBM, and AT are registered trademarks of International Business Machines Corporation.

© 1997 Microchip Technology Inc.

Keyboard Transmission

The keyboard pulls the data line low (start bit) and starts the clock. The eight data bits (least significant bit first) are shifted out, followed by the parity (odd), and stop bit (high). Data is valid after the falling edge of the clock and changes after the rising edge of the clock. If no data is transmitted, both the clock line and data line are high. If the computer pulls the clock line low for at least 60 µs before the tenth bit is transmitted, the keyboard stops the transmission and stores the aborted data in a buffer for retransmission at a later time.

Keyboard Receiving

The computer pulls the data line low (start bit), after which the keyboard starts to shift out 11 clock pulses within 15 ms. Transmission has to be completed within 2 ms. Data from the computer changes after the falling edge of the clock line, and is valid before the rising edge of the clock. After the start bit, eight data bits (least significant bit first), followed by the parity bit (odd), and the stop bit (high) are shifted out by the computer with the clock signal provided by the keyboard. The keyboard pulls the stop bit low in order to acknowledge the receipt of the data. If a transmission error occurs (parity error or similar) the keyboard issues a "RESEND" command to the PC.

Key Pressure Release Encoding

Key pressure is communicated to the PC by sending a scan code. Table 1 lists the scan codes corresponding to keys '0'...'F'. Release is communicated by sending the break code (0F0), followed by the previous scan code.

TABLE 1 SCAN CODES

Codes	Key
45	'0'
16	'1'
1E	'2'
26	'3'
25	'4'
2E	'5'
36	'6'
3D	'7'
3E	'8'
46	'9'
1C	'A'
32	'B'
21	'C'
23	'D'
24	'E'
2B	'F'

© 1997 Microchip Technology Inc.

Proposing a Demo Keypad/Dongle Implementation

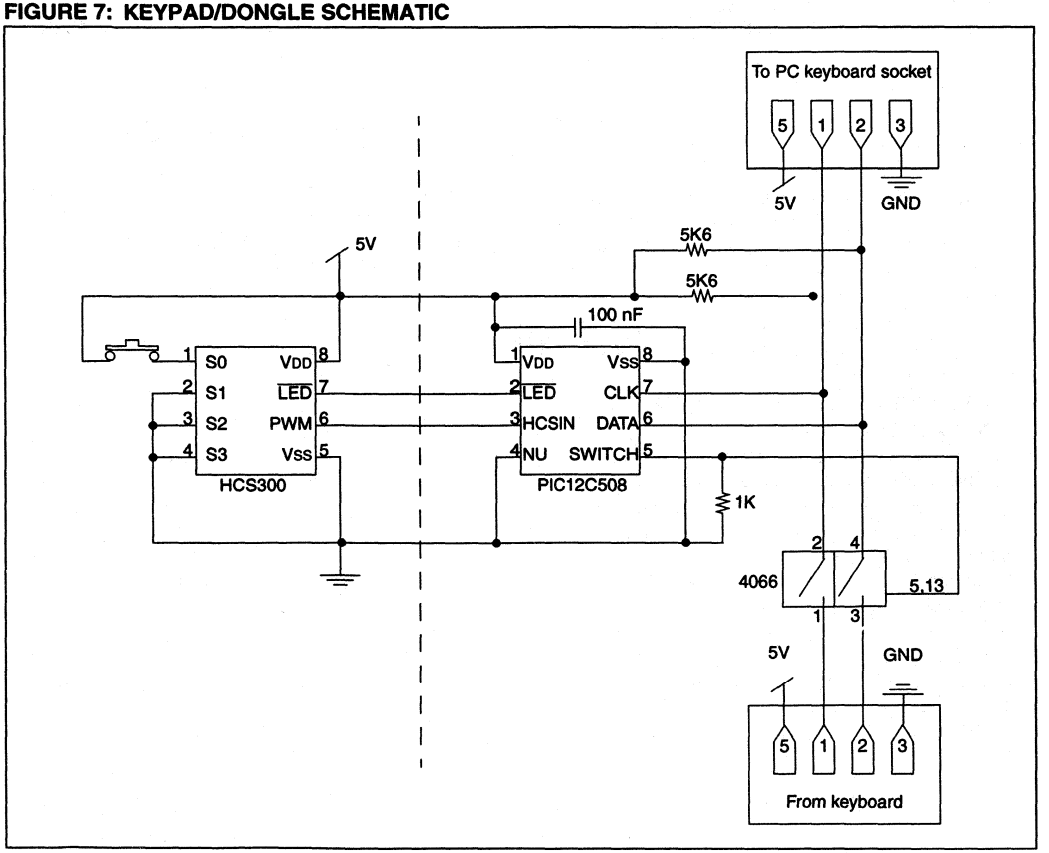
The password generator fits between the keyboard and the PC. A 5-pin plug connects to the PC, supplying power to our device, and the keyboard plugs into the 5-pin socket (Figure 7). The clock and data lines are passed between the PC and keyboard, allowing normal keyboard operations. When S1 is activated, the PIC12C508 receives the new message (16 hex digits) produced by the KEELOQ HCS300 Encoder. The PIC12C508 will then emulate the keyboard, sending the appropriate sequence of key press and key release messages to the PC. To prevent the keyboard from interpreting this transmission as a 'request to send' from the PC, it is necessary to isolate the keyboard from the clock line and data line during the transmission.

The KEELOQ HCS300 Encoder can be part of the dongle or can be removable, like a key, in order to allow different encoders with different encryption keys or serial numbers to be easily exchanged.

Power consumption has to be the lowest possible in order not to excessively load the line. Size and component count should also be kept to the possible minimum in order to allow for a very small package. Ideally, the whole circuit should fit into a small gap between the two connectors.

In the implementation we are proposing, an HCS300 KEELOQ code hopping encoder is used together with a PIC12C508 microcontroller.For simplicity, a standard CMOS quadruple switch (4066) is used to alternatively connect the dongle or the keyboard to the PC line. The HCS300 and the PIC12C508 (both available in 8-pin DIP or SOIC packages), draw extremely low currents as well as internally produce the clock required to operate the dongle. Beside a couple of pull-up resistors required for the clock line and data line, no other components are required to obtain a fully functional hopping password dongle (Figure 7).

FIGURE 7: KEYPAD/DONGLE SCHEMATIC



© 1997 Microchip Technology Inc.

Software Implementation

The software is composed of three short code segments:

- Receive routine for the KEELOQ HCS300 Encoder
- Keyboard Emulation Routines
- Main loop routine.

Receive Routine for the HCS300 Encoder (RECEIVE)

The RECEIVE routine gathers the first 64 bits transmitted by the HCS300, ignoring the last two bits (repeat and battery status) as they carry no useful information for this application, and packs them into a 8-bytes buffer (Buffer0...Buffer7).

Keyboard Emulation Routines (Sendbit, SendKey)

These implement the transmission of the key scan codes according to the IBM-PC/AT keyboard protocol.

Main Loop

While the CMOS switch connects the PC to the keyboard clock and data lines, the $\overline{\text{LED}}$ output line is continuously polled to detect the activation of the HCS300

Note: Any combination of its four input lines after debouncing activates the encoder.

When the $\overline{\text{LED}}$ line goes low, the CMOS switch is activated to isolate the clock and data lines from the keyboard. The RECEIVE routine is called.

Upon successfully receiving a transmission, a loop is entered where 16 hex digits from the receive buffer are transmitted as a sequence of key press and key release messages, separated by appropriate delays repetitively calling the SENDKEY subroutine.

The software has been developed in the simplest possible form and, therefore, is open to a number of optimizations. For example:

- The PIC12C508 could be put to "sleep" to further reduce power consumption.
- The encoder could be removable and its presence/activation should be properly detected.
- For simplicity, the presented RECEIVE routine requires a 400 µs transmission speed being configured in the encoder, while a more flexible multi-baud rate routine can be used as presented in various other application notes.
- A second code word could be compared with the first code word received to recognize transmission errors (although highly improbable when the encoder is wired to the PIC12508), since there is no decryption, there is no other means to tell that the transmission has not been corrupted.

Decoding Options

A code hopping password can be used to validate access to a wide variety of electronic services providing the recipient application (typically, it will be some software running on a server) is capable of following some simple decryption and verification steps. The fixed unencrypted part of the code (last 8 digits) can be used to identify the user (7 digits) and the function activated on the encoder (1 out of 15, corresponding to the last digit).

The hopping part has to be decrypted using the appropriate 64-bit decryption key. Depending on the desired level of security, many different key generation and management techniques can be adopted. For example, the key could be deduced by the User ID and a Manufacturer's Key, by the "seed" code of the encoder or could simply be a fixed 64-bit constant.

Learning techniques can also be applied so that the application actually autonomously acquires the required keys and builds a database of users, ID codes, and decryption keys. For a further analysis, consult the following literature:

AN645	*PIC16C57 Based Code Hopping Security System* (DS00645)
AN662	*KEELOQ Code Hopping Decoder Using Secure Learn* (DS00662)
AN663	*KEELOQ Simple Code Hopping Decoder* (DS00663)
TB001	*An Introduction to KeeLoq Code Hopping* (DS91000A)
TB003	*An Introduction to KEELOQ Code Hopping* (DS91002A).

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: HOPPASW.LST

```
MPASM 01.40 Released          HOPPASW.ASM   2-20-1997  16:45:24         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001     LIST        n=0, c=132
                     00002     PROCESSOR   PIC12C508
                     00003     RADIX       HEX
                     00004
                     00005 ;* filename: HOPPASW.ASM
                     00006 ;*****************************************************************
                     00007 ;*      Author:             Lucio Di Jasio
                     00008 ;*      Company:            Microchip Technology
                     00009 ;*      Revision:           RevA0
                     00010 ;*      Date:               4-sept-96
                     00011 ;*      Assembled using:    MPASM rev. 1.40
                     00012 ;*****************************************************************
                     00013 ;*      include files:
                     00014 ;*          p12C508.inc
                     00015 ;*
                     00016 ;*****************************************************************
                     00017 ;*     HCS300 to keyboard interface, for hopping password generation
                     00018 ;*
                     00019 ;*                 /                 /
                     00020 ;*        HCS300  |5V               |5V
                     00021 ;*      +--------+ |                 | +---------+
                     00022 ;* Key1--+S1      +-+                 +-+Vdd      +-----GND
                     00023 ;* Key2--+S2  LED+----------------------+LED      +-----PCCLK
                     00024 ;* Key3--+S3  PWM+----------------------+HCSIN    +-----DATA
                     00025 ;* Key4--+S4      +-+                  -+nc       +-----SWITCH
                     00026 ;*      +--------+ |                   +---------+
                     00027 ;*                 |
                     00028 ;*                 VGnd
                     00029 ;*
                     00030 ;*****************************************************************
                     00031
                     00032     INCLUDE "\pic\include\p12c508.inc"
                     00001         LIST
                     00002 ; P12C508.INC Standard Header File, Vers 1.01 Microchip Technology. Inc.
                     00103         LIST
                     00033
0FFF 0FEA            00034     __CONFIG _IntRC_OSC & _MCLRE_OFF & _CP_OFF & _WDT_OFF
01FF 0000 0003 0001  00035     __IDLOCS H'0312'
     0002
                     00036
                     00037 ;*****************************************************************
                     00038 ;* internal 4MHz clock
                     00039 ;* internal reset
                     00040 ;* no code protect (?)
                     00041 ;* no watchdog
                     00042 ;* ID code is "0312"
                     00043 ;*****************************************************************
                     00044
                     00045 ;
                     00046 ; pin description
                     00047 ;
                     00048 #define LED         GPIO,5      ; Led from HCS300
```

```
                    00049 #define HCSIN       GPIO,4      ; PWM from HCS300
                    00050 #define NC          GPIO,3      ; not used
                    00051 #define SWITCH      GPIO,2      ; sense clock line to/from PC
                    00052 #define DATA        GPIO,1      ; data to PC
                    00053
                    00054 #define PCCLK       0           ; clock to PC
                    00055
                    00056 #define MASKDEF     b'11111011' ; only SWITCH pin in output
                    00057 #define MASKLOW     b'11111001' ; prepare data low
                    00058 #define MASKHIGH    b'11111011' ; prepare data high
                    00059
                    00060 ;
                    00061 ; RAM assignments
                    00062 ;
                    00063     CBLOCK  07
  00000007          00064         BUFFER0         ; receive buffer for hcs300
  00000008          00065         BUFFER1
  00000009          00066         BUFFER2
  0000000A          00067         BUFFER3
  0000000B          00068         BUFFER4
  0000000C          00069         BUFFER5
  0000000D          00070         BUFFER6
  0000000E          00071         BUFFER7
  0000000F          00072         BITCOUNT        ; counters
  00000010          00073         BYTECOUNT       ;
  00000011          00074         TIMEHI          ; timing
  00000012          00075         TIMELO          ;
  00000013          00076         PARITY          ; transmission parity bit
  00000014          00077         AUX             ; g.p.
  00000015          00078         KEY             ; key to encode
  00000016          00079         GPIOTEMP        ; temp copy of tris register
                    00080     ENDC
                    00081 ;-----------------------------------------------------------------------
                    00082
0000                00083             ORG     0
                    00084
0000                00085 Start
0000 0025           00086             movwf   OSCCAL      ; calibrate
0001 0A7A           00087             goto    Main
                    00088
                    00089 ;-----------------------------------------------------------------------
                    00090 ; keyscan table
                    00091 ;
0002                00092 ScanCode
0002 01E2           00093             addwf   PCL,F
0003 0845           00094             retlw   45          ; key '0'
0004 0816           00095             retlw   16          ; key '1'
0005 081E           00096             retlw   1E          ; key '2'
0006 0826           00097             retlw   26          ; key '3'
0007 0825           00098             retlw   25          ; key '4'
0008 082E           00099             retlw   2E          ; key '5'
0009 0836           00100             retlw   36          ; key '6'
000A 083D           00101             retlw   3D          ; key '7'
000B 083E           00102             retlw   3E          ; key '8'
000C 0846           00103             retlw   46          ; key '9'
000D 081C           00104             retlw   1C          ; key 'A'
000E 0832           00105             retlw   32          ; key 'B'
000F 0821           00106             retlw   21          ; key 'C'
0010 0823           00107             retlw   23          ; key 'D'
0011 0824           00108             retlw   24          ; key 'E'
0012 082B           00109             retlw   2B          ; key 'F'
                    00110
                    00111 #define     BREAK   0F0         ; break scan code
                    00112
                    00113 ;*****************************************************************
                    00114 ;*  SubDelay
```


© 1997 Microchip Technology Inc.

```
                    00115 ;*      short delay functions N us
                    00116 ;*
                    00117 ;*      Input Variables:
                    00118 ;*          none
                    00119 ;*      Output Variables:
                    00120 ;*          none
                    00121 ;************************************************************************
                    00122 ;
0013 0000           00123 SubDelay10    nop
0014 0000           00124 SubDelay9     nop
0015 0000           00125 SubDelay8     nop
0016 0000           00126 SubDelay7     nop
0017 0000           00127 SubDelay6     nop
0018 0000           00128 SubDelay5     nop
0019 0800           00129 SubDelay4     retlw   0           ; 2 call + N nop + 2 retlw
                    00130
                    00131 ;************************************************************************
                    00132 ;*  Wait10ms
                    00133 ;*      waits for approx 10ms
                    00134 ;*
                    00135 ;*      Input Variables:
                    00136 ;*          none
                    00137 ;*      Output Variables:
                    00138 ;*          none
                    00139 ;************************************************************************
                    00140 ;
001A                00141 Wait10ms
001A 0C0F           00142             movlw   .15         ; 15 * .7ms ~= 10ms@4MHz
001B                00143 WaitWx750
001B 0031           00144             movwf   TIMEHI
001C                00145 WaitHi
001C 0072           00146             clrf    TIMELO      ; 256 * 3us ~= 750us@4MHz
001D                00147 WaitLo
001D 02F2           00148             decfsz  TIMELO,F
001E 0A1D           00149             goto    WaitLo
001F 02F1           00150             decfsz  TIMEHI,F
0020 0A1C           00151             goto    WaitHi
0021 0800           00152             retlw   0
                    00153
                    00154 ;************************************************************************
                    00155 ;*  SendBit
                    00156 ;*      sends a bit in AT keyboard protocol
                    00157 ;*
                    00158 ;*      Input Variables:
                    00159 ;*          STATUS,C
                    00160 ;*      Output Variables:
                    00161 ;*          none
                    00162 ;************************************************************************
                    00163 ;
0022                00164 SendBit
0022 0066           00165         clrf    GPIO        ; disable kb and clear out buffers
0023 0CF9           00166         movlw   MASKLOW     ; DATA low prepare
0024 0603           00167         btfsc   STATUS,C
0025 0CFB           00168         movlw   MASKHIGH    ; DATA high prepare
0026 0036           00169         movwf   GPIOTEMP    ; save value
0027 0006           00170         tris    GPIO
                    00171
0028 0C0E           00172         movlw   .14
0029 0032           00173         movwf   TIMELO
002A                00174 SBitT
002A 02F2           00175         decfsz  TIMELO,F        ; 45us loop (data stable)
002B 0A2A           00176         goto    SBitT
                    00177
002C 0416           00178         bcf     GPIOTEMP,PCCLK  ; clk fall
002D 0216           00179         movf    GPIOTEMP,W
002E 0006           00180         tris    GPIO
```


© 1997 Microchip Technology Inc.

```
                  00181
002F 0C0F         00182         movlw   .15
0030 0032         00183         movwf   TIMELO
0031              00184 SBitT2
0031 02F2         00185         decfsz  TIMELO,F        ; 45us loop (data stable)
0032 0A31         00186         goto    SBitT2
                  00187
0033 0516         00188         bsf     GPIOTEMP,PCCLK  ; clk rise
0034 0216         00189         movf    GPIOTEMP,W
0035 0006         00190         tris    GPIO
0036 0800         00191         retlw   0
                  00192
                  00193 ;*************************************************************************
                  00194 ;*  SendKEY
                  00195 ;*      sends a scan code to the PC
                  00196 ;*
                  00197 ;*      Input Variables:
                  00198 ;*          W
                  00199 ;*      Output Variables:
                  00200 ;*          none
                  00201 ;*************************************************************************
                  00202 ;
0037              00203 SendKEY
0037 0034         00204             movwf   AUX         ; temp storage
                  00205
                  00206 ; wait PC ready
0038              00207 SendW
0038 0706         00208             btfss   GPIO,PCCLK  ; test PCCLK
0039 0A38         00209             goto    SendW       ; loop until HIGH
                  00210
                  00211 ; PC request ?
                  00212 ;           btfss   DATA        ; PC pull down Data?
                  00213 ;           goto    RecKEY      ; go receive first
                  00214
                  00215 ; send start bit
003A 0403         00216             bcf     STATUS,C
003B 0922         00217             call    SendBit
                  00218
                  00219 ; than shift out 8 bit LSB first
003C 0C08         00220             movlw   .8
003D 002F         00221             movwf   BITCOUNT
003E 0073         00222             clrf    PARITY
                  00223
003F              00224 SBitL
003F 0334         00225             rrf     AUX,F       ; next bit
0040 0603         00226             btfsc   STATUS,C
0041 02B3         00227             incf    PARITY,F    ; count parity
0042 0922         00228             call    SendBit
0043 02EF         00229             decfsz  BITCOUNT,F
0044 0A3F         00230             goto    SBitL       ; loop for 8 bit
                  00231
0045 02B3         00232             incf    PARITY,F    ; parity odd
0046 0333         00233             rrf     PARITY,F    ; send parity Bit
0047 0922         00234             call    SendBit
                  00235
0048 0503         00236             bsf     STATUS,C
0049 0A22         00237             goto    SendBit     ; send stop bit (high = released)
                  00238
                  00239
                  00240 ;*************************************************************************
                  00241 ;*  Receive
                  00242 ;*      receives first 64 bit transmitted by an HCS300 encoder
                  00243 ;*      simplified to operate with 400us PWM only
                  00244 ;*
                  00245 ;*      Input Variables:
                  00246 ;*          none
```


© 1997 Microchip Technology Inc.

```
                        00247 ;*       Output Variables:
                        00248 ;*           BUFFER0..7
                        00249 ;*************************************************************************
                        00250 ;
004A                    00251 Receive
004A 0686               00252               btfsc   HCSIN           ; wait for a falling edge
004B 0A4A               00253               goto    Receive
                        00254
                        00255 ; will accept sync pulses from  3.0 to 6.1 ms.
                        00256 ; more than 128 cycles but less than 256
                        00257 ; each cycle is 24 us @4MHz
                        00258 ;
  00000038              00259 PREBIT        EQU     .56
                        00260
004C 006F               00261               clrf    BITCOUNT        ; init counter
004D                    00262 Rise
004D 0686               00263               btfsc   HCSIN           ; wait rising edge
004E 0A54               00264               goto    Rise2
                        00265
004F 0913               00266               call    SubDelay10      ; 24us per cycle
0050 0914               00267               call    SubDelay9       ;
                        00268
0051 03EF               00269               incfsz  BITCOUNT,F      ; more than 6,0ms timeout
0052 0A4D               00270               goto    Rise            ; waiting loop
0053 0A4A               00271               goto    Receive         ; timeout restart
                        00272
0054                    00273 Rise2
0054 07EF               00274               btfss   BITCOUNT,7      ; if bit7=1 ok
0055 0A4A               00275               goto    Receive         ; else less than  3.0 ms timeout
                        00276
                        00277 ;-----------------------------------------------------------------------
                        00278 ; read folowing 8 bytes (ignore last 2 bit)
                        00279 ;
0056 0C40               00280               movlw   .64             ; 8 bit per byte
0057 002F               00281               movwf   BITCOUNT        ;
                        00282
0058                    00283 FirstPreload
0058 0C38               00284               movlw   PREBIT          ; first bit needs no balance
0059 0031               00285               movwf   TIMEHI
005A 0A5D               00286               goto    WHL
                        00287
005B                    00288 RNextBit
005B 0C36               00289               movlw   PREBIT-2        ; preload counter
005C 0031               00290               movwf   TIMEHI          ; balance extra rrf time
                        00291
005D                    00292 WHL
005D 02B1               00293               incf    TIMEHI,F        ; measure high period
005E 0643               00294               btfsc   STATUS,Z
005F 0A4A               00295               goto    Receive         ; after 1.2ms (200*6) timeout
0060 0686               00296               btfsc   HCSIN           ; loop while High
0061 0A5D               00297               goto    WHL             ;
                        00298
0062 0C38               00299               movlw   PREBIT          ; preload counter
0063 0032               00300               movwf   TIMELO
                        00301
0064                    00302 WLL
0064 02B2               00303               incf    TIMELO,F        ; measure low period
0065 0643               00304               btfsc   STATUS,Z
0066 0A4A               00305               goto    Receive         ; after 1.2ms (200*6) timeout
0067 0786               00306               btfss   HCSIN           ; loop while Low
0068 0A64               00307               goto    WLL
                        00308
                        00309 ; shift in the new bit
0069 0211               00310               movf    TIMEHI,W
006A 00B2               00311               subwf   TIMELO,F        ; if TIMEHI > TIMELO Carry = 0
006B 032E               00312               rrf     BUFFER7,F       ; insert bit in buffer
```


© 1997 Microchip Technology Inc.

```
006C 032D          00313             rrf     BUFFER6,F
006D 032C          00314             rrf     BUFFER5,F
006E 032B          00315             rrf     BUFFER4,F
006F 032A          00316             rrf     BUFFER3,F
0070 0329          00317             rrf     BUFFER2,F
0071 0328          00318             rrf     BUFFER1,F
0072 0327          00319             rrf     BUFFER0,F
                   00320
                   00321 ; compare duty cycle, to skip preamble
0073 0CE0          00322             movlw   0E0             ; test duty cycle
0074 0152          00323             andwf   TIMELO,W        ; delta >200us? (32 cycles)
0075 0643          00324             btfsc   STATUS,Z
0076 0A4A          00325             goto    Receive         ; no! it's a preamble
                   00326
0077 02EF          00327             decfsz  BITCOUNT,F      ; loop to completion
0078 0A5B          00328             goto    RNextBit
0079 0800          00329             retlw   0
                   00330
                   00331 ;-------------------------------------------------------------------
                   00332 ;*******************************************************************
                   00333 ;*  Main loop
                   00334 ;*      set TRIS and option register
                   00335 ;*      wait for start (LED)
                   00336 ;*      disable keyboard
                   00337 ;*      receive new hopping code
                   00338 ;*      send 16 hex digits
                   00339 ;*      wait transmission end
                   00340 ;*      loop
                   00341 ;*******************************************************************
                   00342 ;
007A               00343 Main
007A 0C04          00344             movlw   b'00000100' ; set switch ON
007B 0026          00345             movwf   GPIO
007C 0CFB          00346             movlw   MASKDEF     ; init port
007D 0006          00347             tris    GPIO
007E 0C00          00348             movlw   0
007F 0002          00349             option
                   00350
0080 06A6          00351             btfsc   LED             ; wait for Led output fall
0081 0A7A          00352             goto    Main
                   00353
0082               00354 Disable
0082 0066          00355             clrf    GPIO            ; send disable kb
                   00356                                     ; SWITCH = LOW
                   00357
0083 094A          00358             call    Receive         ; gets the new hopping code
                   00359
                   00360 ;-------------------------------------------------------------------
                   00361 ; emulate a keyboard and send data as a sequence of 16 key
                   00362 ; pressed and released, one each hex digit
                   00363 ;
0084 0C08          00364             movlw   .8              ; 8 byte from the buffer
0085 0030          00365             movwf   BYTECOUNT
0086 0C07          00366             movlw   BUFFER0         ; init pointer
0087 0024          00367             movwf   FSR
                   00368
0088               00369 KEYL
0088 0200          00370             movf    INDF,W          ; low nibble
0089 0E0F          00371             andlw   0F
008A 0902          00372             call    ScanCode        ; encode hex nibble
008B 0035          00373             movwf   KEY
008C 0937          00374             call    SendKEY         ; emulate key press
008D 091A          00375             call    Wait10ms
008E 0CF0          00376             movlw   BREAK           ; emulate key release
008F 0937          00377             call    SendKEY
0090 0C01          00378             movlw   1
```


© 1997 Microchip Technology Inc.

```
0091 091B           00379             call    WaitWx750       ; wait 750us
0092 0215           00380             movf    KEY,W
0093 0937           00381             call    SendKEY
                    00382
0094 091A           00383             call    Wait10ms
                    00384
0095 0380           00385             swapf   INDF,W          ; high nibble
0096 0E0F           00386             andlw   0F
0097 0902           00387             call    ScanCode        ; encode hex nibble
0098 0035           00388             movwf   KEY
0099 0937           00389             call    SendKEY         ; emulate key press
009A 091A           00390             call    Wait10ms
009B 0CF0           00391             movlw   BREAK           ; emulate key release
009C 0937           00392             call    SendKEY
009D 0C01           00393             movlw   1
009E 091B           00394             call    WaitWx750       ; wait 750us
009F 0215           00395             movf    KEY,W
00A0 0937           00396             call    SendKEY
                    00397
00A1 091A           00398             call    Wait10ms
                    00399
00A2 02A4           00400             incf    FSR,F          ; next byte
00A3 02F0           00401             decfsz  BYTECOUNT,F
00A4 0A88           00402             goto    KEYL
                    00403
                    00404 ;-------------------------------------------------------------------
                    00405 ; now wait for the HCS to stop transmisssion (button release)
                    00406 ;
00A5                00407 Release
00A5 07A6           00408             btfss   LED             ; wait Led rise
00A6 0AA5           00409             goto    Release
00A7 0A7A           00410             goto    Main
                    00411
                    00412             END

MPASM 01.40 Released          HOPPASW.ASM   2-20-1997  16:45:24         PAGE  2

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXX-------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X
0200 : XXX------------- ---------------- ---------------- ----------------
0FC0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   168
Program Memory Words Free:   343


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

SECTION 7
SERIAL EEPROM
APPLICATION NOTES

Communicating with the I²C™ Bus Using the PIC16C5X - AN515 7-1
24C01A Compatibility Issue and Its Mobility for Memory Upgrade - AN517 7-11
Interfacing 93CX6 Serial EEPROMs to PIC16C5X Microcontrollers - AN530 7-13
Logic Powered Serial EEPROMs - AN535 7-35
Basic Serial EEPROM Operation - AN536 7-45
Serial EEPROM Endurance - AN537 7-59
1.8 Volt Technology – Benefits - AN550 7-67
Serial EEPROM Solutions vs. Parallel Solutions - AN551 7-69
Using the 24xx65 and 25xx32 with Stand-alone PIC16C54 Code - AN558 7-73
Optimizing Serial Bus Operations with Proper Write Cycle Times - AN559 7-117
Using the 93LC56 and 93LC66 - AN560 7-121
Using the Microchip Endurance Predictive Software - AN562 7-141
Interfacing the 24LCXXB Serial EEPROMs to the PIC16C54 - AN567 7-145
Questions and Answers Concerning Serial EEPROMs - AN572 7-173
EEPROM Endurance Tutorial - AN601 7-177
How to get 10 Million Cycles Out of Your Microchip Serial EEPROM - AN602 7-181
Converting to 24LCXXB and 93LCXX Serial EEPROMs - AN608 7-183
Interfacing Microchip Serial EEPROMs to Motorola® 68HC11 Microcontroller - AN609 7-185
Using the 24LC21 Dual Mode Serial EEPROM - AN610 7-193
Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI™ Ports - AN613 7-199
Interfacing the 8051 with 2-wire Serial EEPROMs - AN614 7-213
Interfacing the 93XX76 and 93XX86 to a PIC16C5X - AN619 7-223
Interfacing Motorola 68HC11 to Microchip SPI™ Serial EEPROMS - AN646 7-225
Interfacing Microchip PIC16C64/74 to Microchip SPI™ Serial EEPROMs - AN647 7-231
Interfacing Microchip PIC16C54 to Microchip SPI™ Serial EEPROMs - AN648 7-235

7

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

MICROCHIP

AN515

Communicating with the I²C™ Bus Using the PIC16C5X

Author: Bruce Negley
Microchip Technology Inc.

INTRODUCTION

The Microchip Technology Inc.'s 24CXX and 85CXX Serial EEPROMs feature a two wire serial interface bus. The bus protocol is I²C compatible. Interface to a serial port with I²C bus protocol in a microcontroller is trivial. This application note is intended for design engineers who want to develop their software programs to communicate a microcontroller with a 2-wire bus Serial EEPROM through a general purpose I/O port.

Unlike the 3-wire bus Serial EEPROMs, the 24CXX/85CXX communicate with any microcontroller only by a serial data I/O line (SDA) and a serial clock (SCL). Chip select is not required. Data transfer may be initiated only when the bus is not busy. During such transfer, the data line (SDA) must remain stable whenever the clock line (SCL) is high. Changes in the data line while the clock line is high are interpreted as a START or STOP. A typical transfer format is shown in Figure 1.

After the START, a slave address is sent. This address is 7-bits long, the eighth bit is a data direction bit. R/$\overline{W}$ - a logical '0' indicates a transmission WRITE, a logical '1' represents a request for data READ. A data transfer is always terminated by a STOP generated by the master controller. However, if a master still wishes to communicate on the bus, it can generate another START and address another slave without first generating a STOP. Various combinations of read/write formats are then possible within such transfer.

FIGURE 1: TRANSFER FORMAT

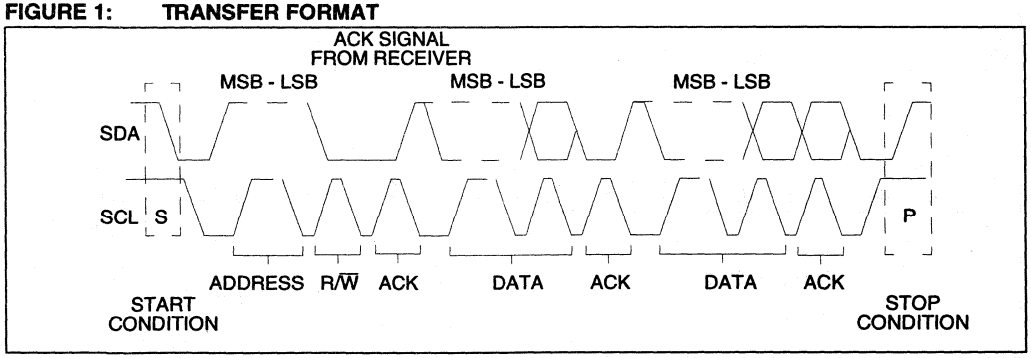
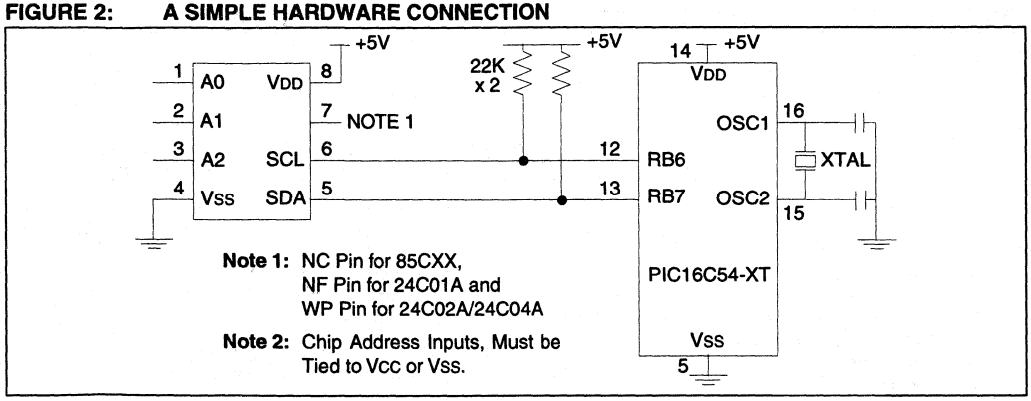


FIGURE 2: A SIMPLE HARDWARE CONNECTION



7

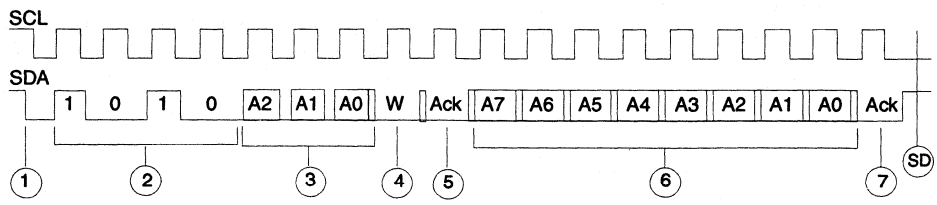
© 1997 Microchip Technology Inc.

An example program has been provided in Appendix A containing all PIC16C54 routines needed to exercise a 24CXX or 85CXX device. A simple hardware connection is illustrated in Figure 2. A maximum of eight 24C01A/24C02A/85C72/85C82's, or four 24C04A/85C92's can be addressed by a microcontroller on the same two wire bus without additional interfaces. Each device is identified by its Chip Address and will only respond to the correct slave address. A detailed bus flow is shown in Figure 3, Figure 4 and Figure 5.

These three figures, as shown below describes how the bit stream is set up for READ and WRITE mode in the microcomputer programming software prior to sending it on the two wire serial bus.

The stop condition, after the write sequence, starts the internal self-timed write cycle which may last up to 6 milliseconds (0.7 ms per byte). Acknowledge signal should be monitored during this period.

FIGURE 3: SETTING THE INTERNAL WORD ADDRESS OF THE 24CXX/85CXX



1. Start Condition
2. 4-Bit Device Code 1010 For 24CXX/85CXX
3. Device Address A0=PA, High Order Address Bit A8 For 24C04A/85C92
4. 0=Write to Address Register
5. EEPROM Drives Bus Low with Acknowledgement (Timeout if No Response)
6. Address for Data, A7-A0 (MSB-LSB)
7. Bus Low for Acknowledgement

FIGURE 4: BYTE WRITE SEQUENCE

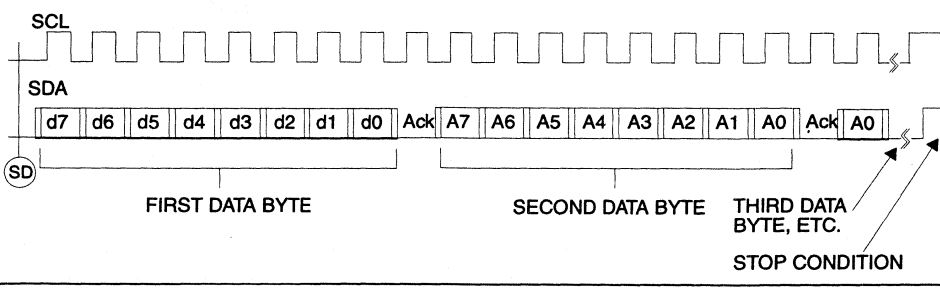
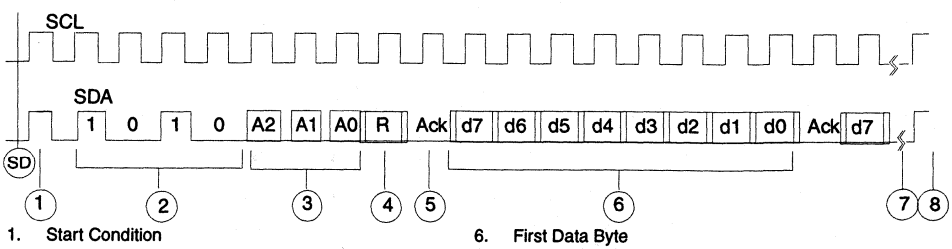


FIGURE 5: READ MODE SEQUENCE



1. Start Condition
2. 4-Bit Device Code 1010 For 24CXX/85CXX
3. Device Address A2, A1, A0. A0=PA, High Order Address Bit A8 For 24C04A/85C92
4. 1=Read Mode
5. Acknowledge EEPROM
6. First Data Byte
7. Second Data Byte, Etc.
8. Stop Condition

Read Up to 128 Bytes (24C01A, 85C72)
Read Up to 256 Bytes (24C02A/04A, 85C82/92)

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com;Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40 Released          I2CBUS.ASM   1-16-1997  12:41:56         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         TITLE " TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x "
                     00002 ;
                     00003         LIST P=16C54
                     00004 ;
                     00005 ;*****************************************************************
                     00006 ;**  Two wire/I2C Bus READ/WRITE Sample Routines of Microchip's
                     00007 ;**  24Cxx / 85Cxx serial CMOS EEPROM interfacing to a
                     00008 ;**  PIC16C54 8-bit CMOS single chip microcomputer
                     00009 ;**  Revsied Version 2.0 (4/2/92).
                     00010 ;**
                     00011 ;**      Part use = PIC16C54-XT/JW
                     00012 ;**  Note: 1) All timings are based on a reference crystal frequency of
                     00013 ;**        2MHz which is equivalent to an instruction cycle time of 2 usec.
                     00014 ;**           2) Address and literal values are read in octal unless
                     00015 ;**              otherwise specified.
                     00016 ;
                     00017 ;
                     00018 ;         Program:          I2CBUS.ASM
                     00019 ;         Revision Date:
                     00020 ;                           1-13-97      Compatibility with MPASMWIN 1.40
                     00021 ;
                     00022 ;*****************************************************************
                     00023 ;
                     00024 ;*****************************************************************
                     00025 ;
                     00026 ;-----------------------------------------------------------------
                     00027 ;         Files Assignment
                     00028 ;-----------------------------------------------------------------
                     00029 ;
00000002             00030 PC        EQU     2               ; Program counter
00000003             00031 STAT      EQU     3               ; PIC status byte
00000004             00032 FSR       EQU     4               ; File Select Register
00000005             00033 RA        EQU     5               ; Port A use to select device address
00000006             00034 RB        EQU     6               ; RB7 = SDA, RB6 = SCL
                     00035 ;
00000008             00036 STATUS    EQU     08              ; Status register
00000009             00037 FLAG      EQU     09              ; Common flag bits register
0000000A             00038 EEPROM    EQU     0A              ; Bit buffer
0000000B             00039 ERCODE    EQU     0B              ; Error code (to indicate bus status)
00000010             00040 ADDR      EQU     10              ; Address register
00000011             00041 DATAI     EQU     11              ; Stored data input register
00000012             00042 DATAO     EQU     12              ; Stored data output register
00000013             00043 SLAVE     EQU     13              ; Device address (1010xxx0)
00000014             00044 TXBUF     EQU     14              ; TX buffer
00000015             00045 RXBUF     EQU     15              ; RX buffer
00000016             00046 COUNT     EQU     16              ; Bit counter
                     00047 ;
00000018             00048 TIMER0    EQU     18              ; Delay timer0
00000019             00049 TIMER1    EQU     19              ; Delay timer1
                     00050 ;
                     00051 ;
                     00052 ;-----------------------------------------------------------------
                     00053 ;                       Bit Assignments
                     00054 ;-----------------------------------------------------------------
```


© 1997 Microchip Technology Inc.

```
                00055 ;
                00056 ; Status bits
                00057 ;
  00000002      00058 Z         EQU     2
  00000000      00059 C         EQU     0
                00060 ;
                00061 ; FLAG Bits
                00062 ;
  00000000      00063 ERR1      EQU     0               ; Error flag
                00064 ;
                00065 ; Instruction Destination Bits
                00066 ;
  00000001      00067 F         EQU     1
  00000000      00068 W         EQU     0
                00069 ;
                00070 ; EEPROM Bits
                00071 ;
  00000007      00072 DI        EQU     7               ; EEPROM input
  00000006      00073 DO        EQU     6               ; EEPROM output
                00074 ;
                00075 ; I2C Device Bits
                00076 ;
  00000007      00077 SDA       EQU     7               ; RB7, data in/out
  00000006      00078 SCL       EQU     6               ; RB6, serial clock
                00079 ;
                00080 ;END FILES/BITS EQUATE
                00081  PAGE
                00082 ;
                00083 ;--------------------------------------------------------------------
                00084 ; Two wire/I2C - CPU communication error status table and subroutine
                00085 ;--------------------------------------------------------------------
                00086 ;  input  :     W-reg   = error code
                00087 ;  output :     ERCODE  = error code
                00088 ;           FLAG(ERR1) = 1
                00089 ;
                00090 ;       code        error status mode
                00091 ;     -------       ------------------------------------------
                00092 ;         1   :     SCL locked low by device (bus is still busy)
                00093 ;         2   :     SDA locked low by device (bus is still busy)
                00094 ;         3   :     No acknowledge from device (no handshake)
                00095 ;         4   :     SDA bus not released for master to generate STOP bit
                00096 ;--------------------------------------------------------------------
                00097 ;
                00098 ;Subroutine to identify the status of the serial clock (SCL) & serial data
                00099 ;(SDA) condition according to the error status table. Codes generated are
                00100 ;useful for bus/device diagnosis.
                00101 ;
0000            00102 ERR
0000 0709       00103         BTFSS   FLAG,ERR1       ; Remain as first error encountered
0001 002B       00104         MOVWF   ERCODE          ; Save error code
0002 0509       00105         BSF     FLAG,ERR1       ; Set error flag
0003 0800       00106         RETLW   0
                00107 ;
                00108 ;--------------------------------------------------------------------
                00109 ;       START bus communication routine
                00110 ;--------------------------------------------------------------------
                00111 ;       input   : none
                00112 ;       output  : initialize bus communication
                00113 ;--------------------------------------------------------------------
                00114 ;
                00115 ;Generate START bit (SCL is high while SDA goes from high to low
                00116 ;transition) and check status of the serial clock.
0004            00117 BSTART
0004 0C3F       00118         MOVLW   B'00111111'     ; Put SCL, SDA line in output state
0005 0006       00119         TRIS    RB
                00120 ;*************************************************************************
```


© 1997 Microchip Technology Inc.

```
0006 05E6       00121         bsf     RB,SDA          ;make sure sda is high
                00122 ;*************************************************************************
0007 05C6       00123         BSF     RB,SCL          ; Set clock high
0008 0C01       00124         MOVLW   1               ; Ready error status code 1
0009 07C6       00125         BTFSS   RB,SCL          ; Locked?
000A 0900       00126         CALL    ERR             ; SCL locked low by device
000B 04E6       00127         BCF     RB,SDA          ; SDA goes low during SCL high
000C 0000       00128         NOP                     ; Timing adjustment
000D 0000       00129         NOP
000E 0000       00130         NOP
000F 04C6       00131         BCF     RB,SCL          ; Start clock train
0010 0800       00132         RETLW   0
                00133 ;
                00134 ;END SUB
                00135  PAGE
                00136 ;
                00137 ;-------------------------------------------------------------------------
                00138 ;       STOP bus communication routine
                00139 ;-------------------------------------------------------------------------
                00140 ;       Input   :       None
                00141 ;       Output  :       Bus communication, STOP condition
                00142 ;-------------------------------------------------------------------------
                00143 ;
                00144 ;Generate STOP bit (SDA goes from low to high during SCL high state)
                00145 ;and check bus conditions.
                00146 ;
0011            00147 BSTOP
                00148 ;*************************************************************************
0011 0C3F       00149         MOVLW   B'00111111'     ; Put SCL, SDA line in output state
0012 0006       00150         TRIS    RB
                00151 ;*************************************************************************
0013 04E6       00152         BCF     RB,SDA          ; Return SDA to low
0014 05C6       00153         BSF     RB,SCL          ; Set SCL high
0015 0000       00154         nop
0016 0000       00155         nop
0017 0000       00156         nop
0018 0C01       00157         MOVLW   1               ; Ready error code 1
0019 07C6       00158         BTFSS   RB,SCL          ; High?
001A 0900       00159         CALL    ERR             ; No, SCL locked low by device
001B 05E6       00160         BSF     RB,SDA          ; SDA goes from low to high during SCL high
001C 0C04       00161         MOVLW   4               ; Ready error code 4
001D 07E6       00162         BTFSS   RB,SDA          ; High?
001E 0900       00163         CALL    ERR             ; No, SDA bus not release for STOP
001F 0800       00164         RETLW   0
                00165 ;
                00166 ;END SUB
                00167 ;
                00168;-------------------------------------------------------------------------
                00169 ;       Serial data send from PIC to serial EEPROM, bit-by-bit subroutine
                00170 ;-------------------------------------------------------------------------
                00171 ;       Input   :       None
                00172 ;       Output  :       To (DI) of serial EEPROM device
                00173 ;-------------------------------------------------------------------------
                00174 ;
0020            00175 BITIN
0020 0CBF       00176         MOVLW   B'10111111'     ; Force SDA line as input
0021 0006       00177         TRIS    RB
0022 05E6       00178         BSF     RB,SDA          ; Set SDA for input
0023 04EA       00179         BCF     EEPROM,DI
0024 05C6       00180         BSF     RB,SCL          ; Clock high
0025 0C01       00181         MOVLW   1
0026 06C6       00182         BTFSC   RB,SCL          ; Skip if SCL  is high
0027 0A2B       00183         GOTO    BIT1
0028 0709       00184         BTFSS   FLAG,ERR1       ; Remain as first error encountered
0029 002B       00185         MOVWF   ERCODE          ; Save error code
002A 0509       00186         BSF     FLAG,ERR1       ; Set error flag
```

7

© 1997 Microchip Technology Inc.

```
002B            00187 BIT1
002B 06E6       00188         BTFSC   RB,SDA          ; Read SDA pin
002C 05EA       00189         BSF     EEPROM,DI       ; DI = 1
002D 0000       00190         NOP                     ; Delay
002E 04C6       00191         BCF     RB,SCL          ; Return SCL to low
002F 0800       00192         RETLW   0
                00193 ;
                00194 ;END SUB
                00195  PAGE
                00196 ;
                00197 ;-------------------------------------------------------------------
                00198 ;   Serial data receive from serial EEPROM to PIC, bit-by-bit subroutine
                00199 ;-------------------------------------------------------------------
                00200 ;       Input   :       EEPROM file
                00201 ;       Output  :       From (DO) of serial EEPROM device to PIC
                00202 ;-------------------------------------------------------------------
                00203 ;
0030            00204 BITOUT
0030 0C3F       00205         MOVLW   B'00111111'     ; Set SDA, SCL as outputs
0031 0006       00206         TRIS    RB
0032 07CA       00207         BTFSS   EEPROM,DO
0033 0A3C       00208         GOTO    BIT0
0034 05E6       00209         BSF     RB,SDA          ; Output bit 0
0035 0C02       00210         MOVLW   2
0036 06E6       00211         BTFSC   RB,SDA          ; Check for error code 2
0037 0A40       00212         GOTO    CLK1
0038 0709       00213         BTFSS   FLAG,ERR1       ; Remain as first error encountered
0039 002B       00214         MOVWF   ERCODE          ; Save error code
003A 0509       00215         BSF     FLAG,ERR1       ; Set error flag
003B 0A40       00216         GOTO    CLK1            ; SDA locked low by device
                00217 ;
003C            00218 BIT0
003C 04E6       00219         BCF     RB,SDA          ; Output bit 0
003D 0000       00220         NOP                     ; Delay
003E 0000       00221         NOP
003F 0000       00222         NOP
0040            00223 CLK1
0040 05C6       00224         BSF     RB,SCL
0041 0C01       00225         MOVLW   1               ; Error code 1
0042 06C6       00226         BTFSC   RB,SCL          ; SCL locked low?
0043 0A47       00227         GOTO    BIT2            ; No.
0044 0709       00228         BTFSS   FLAG,ERR1       ; Yes.
0045 002B       00229         MOVWF   ERCODE          ; Save error code
0046 0509       00230         BSF     FLAG,ERR1       ; Set error flag
0047            00231 BIT2
0047 0000       00232         NOP
0048 0000       00233         NOP
0049 04C6       00234         BCF     RB,SCL          ; Return SCL to low
004A 0800       00235         RETLW   0
                00236 ;
                00237 ;END SUB
                00238  PAGE
                00239 ;
                00240 ;
                00241 ;-------------------------------------------------------------------
                00242 ;       RECEIVE         DATA    subroutine
                00243 ;-------------------------------------------------------------------
                00244 ;       Input   :       None
                00245 ;       Output  :       RXBUF = Receive 8-bit data
                00246 ;-------------------------------------------------------------------
                00247 ;
004B            00248 RX
004B 0C08       00249         MOVLW   .8              ; 8 bits of data
004C 0036       00250         MOVWF   COUNT
004D 0075       00251         CLRF    RXBUF
                00252 ;
```


© 1997 Microchip Technology Inc.

```
004E                00253 RXLP
004E 0375           00254         RLF     RXBUF, F        ; Shift data to buffer
004F 0703           00255         SKPC
0050 0415           00256         BCF     RXBUF,0         ; carry ---> f(0)
0051 0603           00257         SKPNC
0052 0515           00258         BSF     RXBUF,0
0053 0920           00259         CALL    BITIN
0054 06EA           00260         BTFSC   EEPROM,DI
0055 0515           00261         BSF     RXBUF,0         ; Input bit =1
0056 02F6           00262         DECFSZ  COUNT, F        ; 8 bits?
0057 0A4E           00263         GOTO    RXLP
0058 05CA           00264         BSF     EEPROM,DO       ; Set acknowledge bit = 1
0059 0930           00265         CALL    BITOUT          ; to STOP further input
005A 0800           00266         RETLW   0
                    00267 ;
                    00268 ;END SUB
                    00269 ;
                    00270 ;-------------------------------------------------------------------------
                    00271 ;       TRANSMIT        DATA    subroutine
                    00272 ;-------------------------------------------------------------------------
                    00273 ;       Input   :       TXBUF
                    00274 ;       Output  :       Data X'mitted to EEPROM device
                    00275 ;-------------------------------------------------------------------------
                    00276 ;
005B                00277 TX
005B 0C08           00278         MOVLW   .8
005C 0036           00279         MOVWF   COUNT
                    00280 ;
005D                00281 TXLP
005D 04CA           00282         BCF     EEPROM,DO       ; Shift data bit out.
005E 06F4           00283         BTFSC   TXBUF,7         ; If shifted bit = 0, data bit = 0
005F 05CA           00284         BSF     EEPROM,DO       ; Otherwise data bit = 1
0060 0930           00285         CALL    BITOUT          ; Serial data out
0061 0374           00286         RLF     TXBUF, F        ; Rotate TXBUF left
0062 0703           00287         SKPC                    ; f(6) ---> f(7)
0063 0414           00288         BCF     TXBUF,0         ; f(7) ---> carry
0064 0603           00289         SKPNC                   ; carry ---> f(0)
0065 0514           00290         BSF     TXBUF,0
0066 02F6           00291         DECFSZ  COUNT, F        ; 8 bits done?
0067 0A5D           00292         GOTO    TXLP            ; No.
0068 0920           00293         CALL    BITIN           ; Read acknowledge bit
0069 0C03           00294         MOVLW   3
006A 06EA           00295         BTFSC   EEPROM,DI       ; Check for acknowledgement
006B 0900           00296         CALL    ERR             ; No acknowledge from device
006C 0800           00297         RETLW   0
                    00298 ;
                    00299 ;END SUB
                    00300  PAGE
                    00301 ;
                    00302 ;-------------------------------------------------------------------------
                    00303 ;       BYTE-WRITE, write one byte to EEPROM device
                    00304 ;-------------------------------------------------------------------------
                    00305 ;       Input   :       DATAO= data to be written
                    00306 ;                       ADDR    = destination address
                    00307 ;                       SLAVE   = device address (1010xxx0)
                    00308 ;       Output  :       Data written to EEPROM device
                    00309 ;-------------------------------------------------------------------------
                    00310 ;
0080                00311         ORG     080             ; The location for BYTE-WRITE routine can be
                    00312 ;                               ; assigned anywhere between (377-777) octal.
0080                00313 WRBYTE
0080 0213           00314         MOVF    SLAVE,W         ; Get SLAVE address
0081 0034           00315         MOVWF   TXBUF           ; to TX buffer
0082 0904           00316         CALL    BSTART          ; Generate START bit
0083 095B           00317         CALL    TX              ; Output SLAVE address
0084 0210           00318         MOVF    ADDR,W          ; Get WORD address
```

7

```
0085 0034       00319         MOVWF   TXBUF           ; into buffer
0086 095B       00320         CALL    TX              ; Output WORD address
0087 0212       00321         MOVF    DATAO,W         ; Move DATA
0088 0034       00322         MOVWF   TXBUF           ; into buffer
0089 095B       00323         CALL    TX              ; Output DATA and detect acknowledgement
008A 0911       00324         CALL    BSTOP           ; Generate STOP bit
008B 0AD8       00325         goto    wrt_end
                00326 ;
                00327 ;
                00328 ;
                00329 ;-------------------------------------------------------------------------
                00330 ;       BYTE-READ, read one byte from serial EEPROM device
                00331 ;-------------------------------------------------------------------------
                00332 ;       Input   :       ADDR    = source address
                00333 ;                       SLAVE   = device address (1010xxx0)
                00334 ;       Output  :       DATAI   = data read from serial EEPROM
                00335 ;-------------------------------------------------------------------------
                00336 ;
00C0            00337         ORG     0C0             ; The location for BYTE-READ routine can be
                00338 ;                               ; assigned anywhere between (377-777) octal.
00C0            00339 RDBYTE
00C0 0213       00340         MOVF    SLAVE,W         ; Move SLAVE address
00C1 0034       00341         MOVWF   TXBUF           ; into buffer (R/W = 0)
00C2 0904       00342         CALL    BSTART          ; Generate START bit
00C3 095B       00343         CALL    TX              ; Output SLAVE address. Check ACK.
00C4 0210       00344         MOVF    ADDR,W          ; Get WORD address
00C5 0034       00345         MOVWF   TXBUF
00C6 095B       00346         CALL    TX              ; Output WORD address. Check ACK.
00C7 0904       00347         CALL    BSTART          ; START READ (if only one device is
00C8 0213       00348         MOVF    SLAVE,W         ; connected to the I2C bus)
00C9 0034       00349         MOVWF   TXBUF
00CA 0514       00350         BSF     TXBUF,0         ; Specify READ mode (R/W = 1)
00CB 095B       00351         CALL    TX              ; Output SLAVE address
00CC 094B       00352         CALL    RX              ; READ in data and acknowledge
00CD 0911       00353         CALL    BSTOP           ; Generate STOP bit
00CE 0215       00354         MOVF    RXBUF,W         ; Save data from buffer
00CF 0031       00355         MOVWF   DATAI           ; to DATAI file.
00D0 0ADA       00356         goto    rd_end
                00357 ;
                00358 ;Test program to read and write ramdom
00D1            00359 start
00D1 0CAE       00360         movlw   0AE             ;set A2=A1=A0=1
00D2 0033       00361         movwf   SLAVE           ;       /
00D3 0C02       00362         movlw   2               ;set r/w loc. = 2
00D4 0030       00363         movwf   ADDR            ;       /
00D5 0C55       00364         movlw   55              ;write 55 to SEEPROM
00D6 0032       00365         movwf   DATAO           ;       /
00D7 0A80       00366         goto    WRBYTE          ;write a byte
00D8            00367 wrt_end
00D8 09DF       00368         call    delay           ;wait for write
                00369                                 ;operation (internal)
00D9 0AC0       00370         goto    RDBYTE          ;read back data
00DA            00371 rd_end
00DA 0C55       00372         movlw   55              ;test if read
00DB 0191       00373         xorwf   DATAI,W         ;correct?
00DC 0743       00374         btfss   STAT,Z          ;yes then skip
00DD            00375 wrong
00DD 0ADD       00376         goto    wrong
00DE            00377 correct
00DE 0ADE       00378         goto    correct
                00379 ;
                00380 ;At 2.0Mhz, delay = approx. 3mS.
00DF            00381 delay
00DF 007F       00382         clrf    1F              ;clear last location
00E0            00383 dly1
00E0 0000       00384         nop
```


© 1997 Microchip Technology Inc.

```
00E1 0000       00385         nop
00E2 0000       00386         nop
00E3 02FF       00387         decfsz  1F, F           ;reduce count
00E4 0AE0       00388         goto    dly1            ;loop
00E5 0800       00389         retlw   0
                00390 ;
01FF            00391         org     0x1FF
01FF 0AD1       00392         goto    start
                00393 ;
                00394         END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)


0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXX--- ----------------
0080 : XXXXXXXXXXXX---- ---------------- ---------------- ----------------
00C0 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXX---------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   160
Program Memory Words Free:   352


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN517

24C01A Compatibility Issue and Its Mobility for Memory Upgrade

Author: *Bruce Negley*
Microchip Technology Inc.

INTRODUCTION

The 24C01A is a 1K (128 x 8) Serial EEPROM which is currently offered by Microchip and Xicor. There are several important differences between the two devices which are discussed in this report. This report refers to the Microchip part as the 24C01A and the Xicor part as the X24C01. It is intended to assist in designing a memory subsystem which is compatible with either device.

COMPATIBILITY ISSUES

There are three major differences between Microchip's 24C01A and Xicor's X24C01 as detailed below.

PAGE MODE DIFFERENCES

The 24C01A was originally designed to work in the same socket as the PCD8572 which has a two-byte page mode. Therefore, its page buffer is two bytes deep. The X24C01 has a page mode of four bytes depth.

If more than two bytes are transmitted to the 24C01A during a page programming cycle, the 24C01A will terminate the write cycle.

In many applications where serial EEPROMs are used and speed is not a key issue, the byte write mode can be used without any loss of system performance. If only the byte write mode is used, there is no compatibility problem (other than slave address software differences discussed in the Software Differences section).

If the page write feature must be used, two different page mode algorithms can be transmitted by the master depending upon whose device is being used. The master will have to first do a polling routine to determine if it is interfacing with a 24C01A or X24C01. This polling technique is discussed in the Software Difference section.

Interestingly, the 24C01A actually updates faster in the page mode even though it has one-half the page depth of the X24C01. This is due to the faster write cycle time of the 24C01A. The two devices are compared in Figure 1.

SOFTWARE DIFFERENCES

Microchip's 24C01A is designed to share a 2-wire bus on which it resides with other devices. To support this, the first byte of each command sequence from the master to the 24C01A must be a slave address. The 24C01A monitors the 2-wire bus for its slave address and "wake-up" from standby mode if the address transmitted matches its address as defined by the voltage level (Vss or Vcc) on pins 1, 2 and 3. X24C01 does not support a multiple device bus and will always "wake-up" if a start condition is detected.

A slave address must be transmitted to the 24C01A at certain points during reading and writing. This slave address is not required by the X24C01. Transmitting a slave address to X24C01 will result in erroneous operation. This problem can be solved by having the master transmit the proper serial bit pattern to the slave, but first the master has to ascertain with which 24C01A it is communicating.

7

FIGURE 1: PAGE MODE DIFFERENCES

	Microchip	Xicor
Max byte program time	1 ms	10 ms
Max page program time	2 ms (2 bytes)	10 ms (4 bytes)
Max time to program 4 bytes	4 ms	10 ms
Max time to rewrite device	128 ms	320 ms

© 1997 Microchip Technology Inc.

The polling consists of the pattern like the one shown below:

| SDA LINE: | START BIT | 00000001 | ACKNOWLEDGE BIT | DATA 7...0 |

The master can do a simple polling routine before beginning serial communication with 24C01A or X24C01 to determine with which device it is working. The proper serial protocol for both devices must be contained in the master controller's firmware. Once the master knows which 24C01A is on the bus, it can execute the proper serial commands.

If an X24C01 is used on the 2-wire bus, an acknowledge bit and eight data bits will be returned whereas 24C01A will issue no response and will ignore the command.

HARDWARE DIFFERENCES

Unlike the X24C01, the 24C01A is designed to share a 2-wire bus with other devices. Chip address bits are included in the slave address for the 24C01A, and are incorporated into pins 1, 2 and 3 of the device. They must be connected to VCC or VSS for proper operation. Since pins 1, 2 and 3 of the Xicor part are NC (no connect) pins and they are not internally connected, they can be tied high or low.

Another hardware difference involves pin 7 which MUST be connected to VSS on the X24C01. The 24C01A can have pin 7 connected to VSS or VCC.

If only one device is planned for the 2-wire bus, the board can be made compatible for either device by connecting pins 1, 2 and 3 to either VSS or VCC and tying pin 7 to VSS.

Mobility For Memory Upgrade And Expansion

In system applications where the master device needs to address more than one serial EEPROM on a 2-wire bus, the Microchip 24C01A offers the flexibility. Up to eight 24C01A's can be connected to the 2-wire bus. More than one Xicor X24C01 connected to the bus may result in bus contention.

If memory upgrade is required, the Microchip 24C01A can be upgraded to the 24C02A (256 x 8) or the 24C04A (512 x 8) in the same IC socket with NO change in hardware. Using the Xicor X24C01, both software and hardware would have to be reconfigured to accommodate the changes.

© 1997 Microchip Technology Inc.

MICROCHIP

AN530

Interfacing 93CX6 Serial EEPROMs to PIC16C5X Microcontrollers

Authors: Stan D'Souza
Microchip Technology Inc.
Bob Ward
Microchip Technology Inc.

INTRODUCTION

Microchip Technology Inc.'s popular 93C46/56/66 and 93LC46/56/66 Serial EEPROMs feature a three/four wire serial interface bus. The attractive price and simple interface make it the ideal device for additional memory space. This application note is intended for design engineers who wish to incorporate a pre-packaged serial EEPROM interface driver into their application.

THE HARDWARE CONNECTION

A typical 4-wire hardware connection is illustrated in Figure 1 and a typical 3-wire connection is illustrated in Figure 2. Since all I/O ports on the PIC16C5X are configurable as input and/or output, a 3-wire interface makes optimum utilization of the I/O pins by having a common connection for the DI and DO lines of the Serial EEPROM. The port pin on the PIC16C5X connected to these pins, has a default setting as an output and is configured, when needed, as an input during program execution.

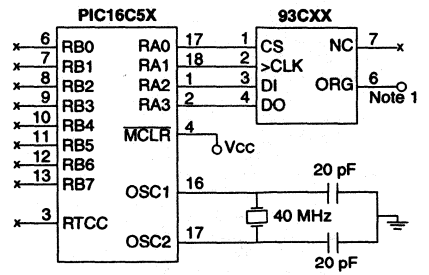
THE SOFTWARE CONNECTION

An example interface driver is listed in Appendix A. A flow diagram is given in Figure 2. The interface driver is written to minimize both overhead to the calling program as well as the program space necessary for its inclusion into the user's code. The driver has been written as a generic driver to service all 93 Series Serial EEPROMs made by Microchip. Processor resources which must be made available to the driver prior to being called are:

1. Two levels of processor stack.
2. Six register locations (four for command/data passing and two for software counters).
3. The File Select Register (FSR), which is used to pass a command/data string pointer to the driver.

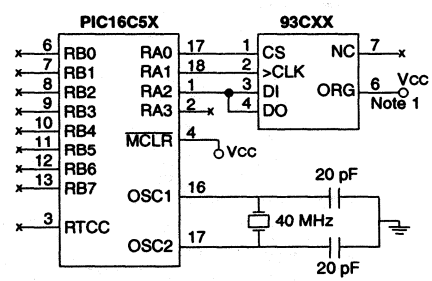
Note: The four command/data passing registers have to be defined consecutively in order for the FSR to access them successfully in the program execution.

FIGURE 1: 4-WIRE CONNECTION



Note 1: FOR 93LCXX
For 16-bit data word connect to Vcc
For 8-bit data word connect to GND

FIGURE 2: 3-WIRE CONNECTION



Note 1: FOR 93LCXX
For 16-bit data word connect to Vcc
For 8-bit data word connect to GND

The user should take the following steps when using the routines provided in Appendix A.

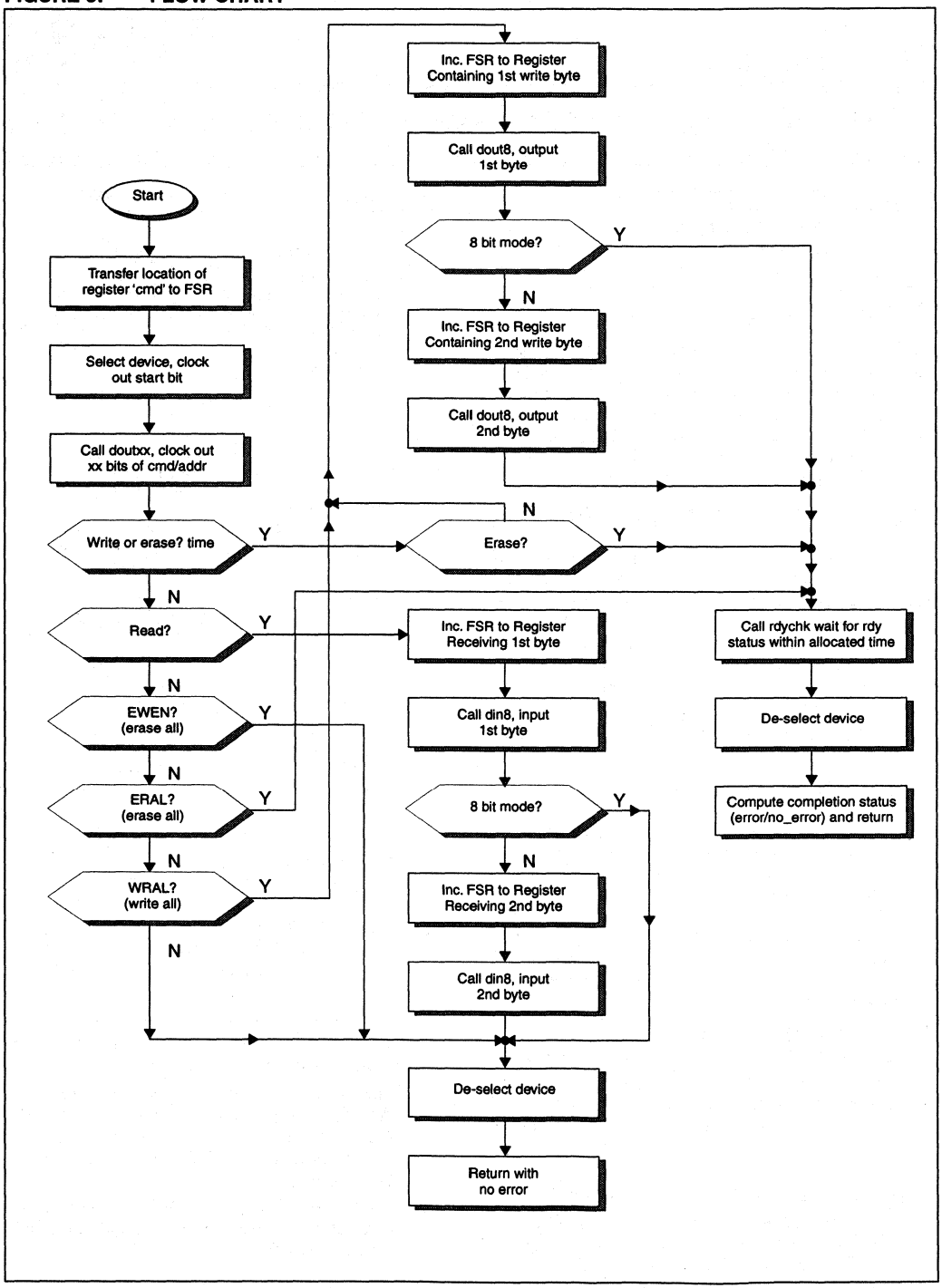
A) Specify and define a 3-/4-wire interface by defining the common connection to the DI/DO lines and setting the equate 'wire3' TRUE or FALSE (4-wire is automatically assumed if 3-wire is false).

B) Specify and define if 16-bit or 8-bit data organization is used, by setting equate 'org8' TRUE or FALSE.

7

© 1997 Microchip Technology Inc.

FIGURE 3: FLOW CHART



© 1997 Microchip Technology Inc.

C) The user should assemble the source file by specifying which type of serial EEPROM is being used. This is done by defining the equate S93C46, S93LC46, etc., as TRUE. Only one device can be TRUE, the rest have to be defined FALSE.

D) The user would invoke the driver as follows:

1. Load the register defined as 'cmd' with the 93CX6 Command Opcode (only the four upper bits are used in this register; see Figure 3).
2. If necessary, load the register defined 'addr' with the lower 8/7/6 bit address of the location.
3. If necessary, load the 9th bit of the address as bit 3 of the register defined as 'cmd' (Figure 3).

Note: READ, WRITE and ERASE commands need to have an address associated with the command and the 9th bit of the address is only required when using the 93C56/66 or 93LC56/66 devices in the 8 bit mode (ORG tied to GND).

4. If necessary, load the register defined as 'highb' and 'lowb' with the 16 bits of data, the most significant byte loaded in 'highb'. In 8 bit mode, 'highb' should be loaded with the 8 bit data.

Note: Only the WRITE and WRAL commands require data to be loaded in the 'highb', 'lowb' locations.

5. Call the driver sub-routine 'see'.
6. Upon completion, the driver will return a completion status in the W register (error/no error). Only commands requiring a status check are capable of returning a valid error/no error status, in all other cases a no error is returned.
7. If the READ command is executed, the 16/8 bit data will be loaded in the 'highb' and 'lowb' registers, where 'highb' contains the MSB in the 16 bit mode and 8 bit data in the 8-bit mode.
   The Example interface assumes a 4 MHz oscillator clock which gives us a 1 µS instruction cycle. If a higher clock speed is used, additional NOPs have to be included in the code in order to meet the minimum clock speed requirements of the 93 Series Serial EEPROMs (see data sheet for further details).

Listing in Appendix B is for an interface to 93C46 Serial EEPROM only.

SUMMARY

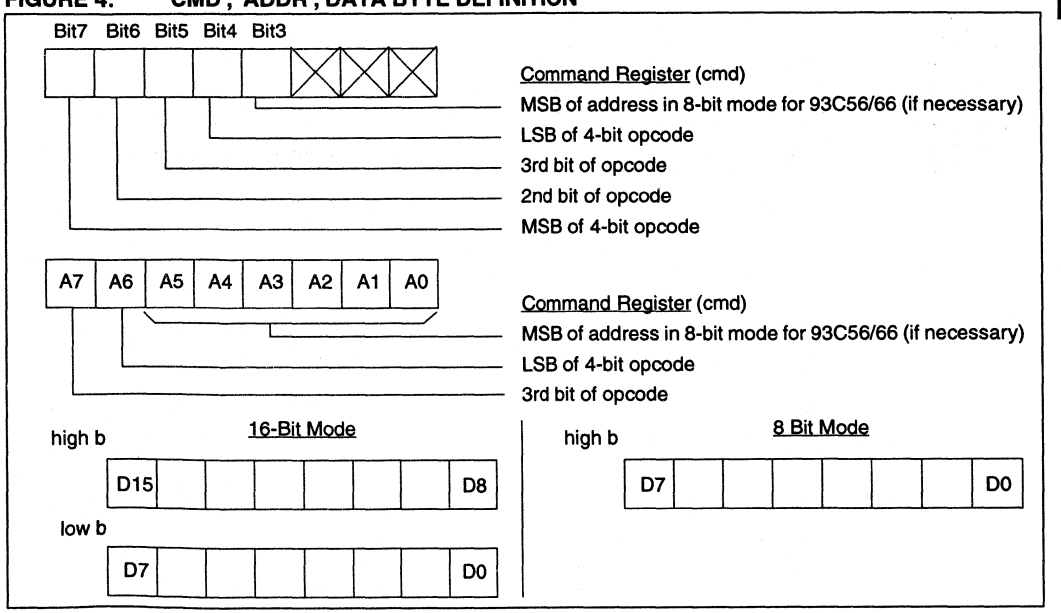
The 93 Series Serial EEPROMs are a simple and versatile method of increasing read/write memory capability in a PIC16C5X application. The 'generic' code in Appendix A makes it easy to incorporate in any PIC16C5X application. Any of the 93 Series Serial EEPROMs can be applied, while at the same time using a minimal amount of I/O, code and RAM resources.

Code size: 6 bytes of RAM.

Appendix A listing:	127 bytes program code (max.) 100 bytes program code (min.)
Appendix B listing:	86 bytes program code

7

FIGURE 4: 'CMD', 'ADDR', DATA BYTE DEFINITION



© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: 93XCXX.ASM

```
MPASM 01.40 Released          93XCXX.ASM   1-16-1997  13:13:47         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001                TITLE        "R/W EEPROM"
                     00002      LIST  P = 16C54
                     00003 ;Serial Eprom interface to PIC16C5X.
                     00004 ;
                     00005 ;
                     00006 ;       Program:          93XCXX.ASM
                     00007 ;       Revision Date:    8/26/92
                     00008 ;                         10/31/92      Comments clarified  REV 2.0
                     00009 ;                         1-13-97       Compatibility with MPASMWIN 1.40
                     00010 ;
                     00011 ;*****************************************************************************
                     00012 ;
                     00013 ;Define Equates:
                     00014 ;
  000001FF           00015 PIC54   EQU     1FFH
                     00016 ;
                     00017     PAGE
                     00018 ;
0000                 00019         ORG     0
0000                 00020 START
0000 0A7B            00021         goto    main            ;run test program
                     00022 ;
                     00023     PAGE
                     00024 ;
  00000001           00025 TRUE    EQU     1
  00000000           00026 FALSE   EQU     0
  00000001           00027 F       EQU     1
  00000000           00028 S93C46  EQU     FALSE
  00000000           00029 S93LC46 EQU     FALSE
  00000000           00030 S93C56  EQU     FALSE
  00000000           00031 S93LC56 EQU     FALSE
  00000001           00032 S93C66  EQU     TRUE
  00000000           00033 S93LC66 EQU     FALSE
  00000001           00034 wire3   equ     TRUE            ; for four-wire setup equate to FALSE
  00000000           00035 org8    EQU     FALSE
                     00036 ;
                     00037         IF      S93LC46
                     00038         IF      org8
                     00039 LC468   EQU     TRUE
                     00040 XC46    EQU     FALSE
                     00041 H16     EQU     FALSE
                     00042 H8      EQU     FALSE
                     00043         ELSE
                     00044 LC468   EQU     FALSE
                     00045 XC46    EQU     TRUE
                     00046 H16     EQU     FALSE
                     00047 H8      EQU     FALSE
                     00048         ENDIF
                     00049     ELSE
                     00050     ENDIF
                     00051     IF      S93C46
                     00052 LC468   EQU     FALSE
                     00053 XC46    EQU     TRUE
                     00054 H16     EQU     FALSE
```


© 1997 Microchip Technology Inc.

```
                00055 H8        EQU     FALSE
                00056       ENDIF
                00057       IF      S93C56 + S93C66 + S93LC56 + S93LC66
                00058           IF      org8
                00059 H8        EQU     TRUE
                00060 H16       EQU     FALSE
                00061 LC468     EQU     FALSE
                00062 XC46      EQU     FALSE
                00063           ELSE
00000001        00064 H16       EQU     TRUE
00000000        00065 H8        EQU     FALSE
00000000        00066 LC468     EQU     FALSE
00000000        00067 XC46      EQU     FALSE
                00068           ENDIF
                00069       ELSE
                00070       ENDIF
                00071
                00072
                00073
                00074 ;*****************************************************************
                00075 ;*                       Register Assignments                    *
                00076 ;*****************************************************************
                00077
00000000        00078 indir   equ     0       ;Use this register as source/destination
                00079                         ;for indirect addressing.
00000002        00080 pc      equ     2       ;PIC Program Counter.
00000003        00081 status  equ     3       ;PIC Status Register.
00000004        00082 fsr     equ     4       ;File Select Register.
00000005        00083 serial  equ     5       ;Port used for 93CX6 control.
                00084                         ;The following four registers must be
                00085                         ;located consecutively in memory.
0000001A        00086 cmd     equ     1a      ;This register contains the 4 bit
                00087                         ;command op code for 93CX6 as follows:
                00088                         ;bit 7 msb of command op code
                00089                         ;bit 6 next bit of op code
                00090                         ;bit 5 next bit of op code
                00091                         ;bit 4 lsb of op code
                00092                         ;bit 3 A8 of address in case of
                00093                         ;56/66 in 8 bit mode.
0000001B        00094 addr    equ     1b      ;memory address of lower 7/8 bits
0000001C        00095 highb   equ     1c      ;Used in read/write routines to store the
                00096                         ;upper byte of a 16 bit data word,
                00097                         ;or the data in a 8 bit data word
0000001D        00098 lowb    equ     1d      ;Used in read/write routines to store the
                00099                         ;lower byte of a 16 bit data word,
                00100                         ;or not used in 8 bit data word.
                00101
0000001E        00102 cnthi   equ     1e      ;Used as the upper byte of a sixteen bit
                00103                         ;loop counter in RDYCHK routine.
0000001F        00104 cnt     equ     1f      ;Used as the lower byte of a sixteen bit
                00105                         ;loop counter in RDYCHK routine, and
                00106                         ;elswhere as an eight bit counter.
0000001E        00107 temp_cmd equ    1e      ;doubles as a temp register for cmd
0000001F        00108 temp_addr equ   1f      ;doubles as a temp register for addr
                00109
                00110 ;*****************************************************************
                00111 ;* Bit Assignments: The following assignments are for 3-wire. For
                00112 ;* 4-wire assign din and dout to seperate pins.
                00113
00000000        00114 carry   equ     0               ;Carry Flag of Status Register.
00000002        00115 zflag   equ     2               ;Zero Flag of Status Register.
                00116
00000002        00117 cs      equ     2               ;Port pin tied to CS on 93CX6.
00000001        00118 din     equ     1               ;Port pin tied to DI on 93CX6. 3-wire
00000001        00119 dout    equ     1               ;Port pin tied to DO on 93CX6. 3-wire
00000003        00120 clock   equ     3               ;Port pin tied to CLK on 93CX6.
```


© 1997 Microchip Technology Inc.

```
                        00121
                        00122 ;*******************************************************************
                        00123 ;*                       General Assignments                       *
                        00124 ;*******************************************************************
                        00125
00000000                00126 no_err  equ     0
00000001                00127 err1    equ     1
00000020                00128 tries   equ     20      ;After issuing a WRITE, ERASE, ERAL, or WRAL
                        00129                         ;command, the approximate number of machine
                        00130                         ;cycles X 256 to wait for the RDY status.
                        00131                         ;This value must be adjusted for operating
                        00132                         ;frequencies other than 4 MHz.
                        00133
00000080                00134 read    equ     b'10000000'     ;read command op code
00000040                00135 write   equ     b'01000000'     ;write command op code
000000C0                00136 erase   equ     b'11000000'     ;erase command op code
00000030                00137 ewen    equ     b'00110000'     ;erase enable command op code
00000000                00138 ewds    equ     b'00000000'     ;erase disable command op code
00000020                00139 eral    equ     b'00100000'     ;erase all command op code
00000010                00140 wral    equ     b'00010000'     ;write all command op code
                        00141
                        00142 ;*******************************************************************
                        00143 ;*                       Macro Definitions                         *
                        00144 ;*******************************************************************
                        00145
                        00146 sel     MACRO                   ;Selects the 93CX6 device.
                        00147         bsf     serial,cs       ;Chip Select (CS) = '1' to select
                        00148         ENDM                    ;the device
                        00149
                        00150 dsel    MACRO                   ;De-select the 93CX6 device.
                        00151         bcf     serial,cs       ;Chip Select (CS) = '0' to de-select
                        00152                                 ;the device.
                        00153         ENDM
                        00154
                        00155 strtbt  MACRO                   ;Issue the Start Bit to the 93CX6.
                        00156         bsf     serial,din      ;Start Bit = '1'.
                        00157         clkit                   ;Clock it out.
                        00158         ENDM
                        00159
                        00160 clkit   MACRO                   ;Clocks a serial data bit into or out
                        00161                                 ;of the 93CX6 device.
                        00162         bsf     serial,clock    ;Clock (CLK) = '1'.
                        00163
                        00164         nop             ;Adjust the number of nop instructions
                        00165                         ;between the assertion and de-assertion of
                        00166                         ;CLK in proportion to the PIC operating
                        00167                         ;frequency.  Refer to the 93CX6 data for the
                        00168                         ;minimum CLK period.
                        00169
                        00170         bcf     serial,clock    ;Clock (CLK) = '0'.
                        00171         ENDM
                        00172 ;
                        00173     PAGE
                        00174 ;*******************************************************************
                        00175 ;*                          DOUTx                                  *
                        00176 ;*******************************************************************
                        00177 ;doutxx, outputs up to 11 bits of op code/data, depending on whether
                        00178 ;a 46/56/66 serial eeprom is being used. The number of bits over 8 are
                        00179 ;saved in the cmd register and the rest in the addr register. Before
                        00180 ;calling this routine the cmd and the addr registers should be loaded
                        00181 ;as follows:
                        00182 ;cmd reg.bits 7|6|5|4|3|2|1|0
                        00183 ;--------------|-|-|-|-|-|-|-|
                        00184 ;              X|X|X|X|X|X|X|Y| --> not used
                        00185 ;              X|X|X|X|X|X|Y|X| --> mot used
                        00186 ;              X|X|X|X|X|Y|X|X| --> not used
```


© 1997 Microchip Technology Inc.

```
                    00187 ;             X|X|X|X|Y|X|X|X|  --> 9th bit of address if necessary
                    00188 ;             X|X|X|Y|X|X|X|X|  --> lsb of command op code
                    00189 ;             X|X|Y|X|X|X|X|X|  --> 3rd bit of command op code
                    00190 ;             X|Y|X|X|X|X|X|X|  --> 2nd bit of command op code
                    00191 ;             Y|X|X|X|X|X|X|X|  --> msb of command op code
                    00192 ;
                    00193 ;addr reg. 6/7/8 bits of address if necessary.
                    00194
                    00195     IF      H8
                    00196 dout11
                    00197         bcf     serial,din
                    00198         rlf     indir           ;rotate thru carry
                    00199         btfsc   status,carry    ;set?
                    00200         bsf     serial,din      ;yes, set output to 1
                    00201         clkit                   ;clk data
                    00202     ENDIF
                    00203     IF      H16+H8
0001                00204 dout10
0001 0360           00205         rlf     indir, F        ;rotate thru carry
0002 0425           00206         bcf     serial,din      ;set output to 0
0003 0603           00207         btfsc   status,carry    ;set?
0004 0525           00208         bsf     serial,din      ;else set output to 1
                    00209         clkit                   ;clk data
                        M                                 ;of the 93CX6 device.
0005 0565               M         bsf     serial,clock    ;Clock (CLK) = '1'.
                        M
0006 0000               M         nop             ;Adjust the number of nop instructions
                        M                         ;between the assertion and de-assertion of
                        M                         ;CLK in proportion to the PIC operating
                        M                         ;frequency.  Refer to the 93CX6 data for the
                        M                         ;minimum CLK period.
                        M
0007 0465               M         bcf     serial,clock    ;Clock (CLK) = '0'.
                    00210     ENDIF
                    00211     IF      H8+H16+LC468
0008                00212 dout9
0008 0360           00213         rlf     indir, F        ;rotate thru carry
0009 0425           00214         bcf     serial,din      ;set output to 0
000A 0603           00215         btfsc   status,carry    ;set?
000B 0525           00216         bsf     serial,din      ;else set output to 1
                    00217         clkit                   ;clk data
                        M                                 ;of the 93CX6 device.
000C 0565               M         bsf     serial,clock    ;Clock (CLK) = '1'.
                        M
000D 0000               M         nop             ;Adjust the number of nop instructions
                        M                         ;between the assertion and de-assertion of
                        M                         ;CLK in proportion to the PIC operating
                        M                         ;frequency.  Refer to the 93CX6 data for the
                        M                         ;minimum CLK period.
                        M
000E 0465               M         bcf     serial,clock    ;Clock (CLK) = '0'.
000F 02A4           00218         incf    fsr, F          ;inc pointer
                    00219     ENDIF
0010                00220 dout8
0010 0C08           00221         movlw   8               ;Initialize loop counter.
0011 003F           00222         movwf   cnt
                    00223 ;
0012 0425           00224 d_o_8   bcf     serial,din      ;Assume that the bit to be transfered
                    00225                                 ;is a '0'.  Hence, de-assert DI.
0013 0360           00226         rlf     indir, F        ;Rotate the actual bit to be
                    00227                                 ;transferred into the carry bit.
0014 0603           00228         btfsc   status,carry    ;Test the carry, if our assumption was
                    00229                                 ;correct, skip the next instruction.
0015 0525           00230         bsf     serial,din      ;No, actual bit was a '1'.  Assert DI.
                    00231         clkit                   ;Clock the 93CX6.
                        M                                 ;of the 93CX6 device.
```


© 1997 Microchip Technology Inc.

```
0016 0565          M         bsf    serial,clock ;Clock (CLK) = '1'.
                   M
0017 0000          M         nop                 ;Adjust the number of nop instructions
                   M                             ;between the assertion and de-assertion of
                   M                             ;CLK in proportion to the PIC operating
                   M                             ;frequency.  Refer to the 93CX6 data for the
                   M                             ;minimum CLK period.
                   M
0018 0465          M         bcf    serial,clock ;Clock (CLK) = '0'.
0019 02FF       00232        decfsz cnt, F       ;Repeat until cnt = 0.
001A 0A12       00233        goto   d_o_8        ;Cnt still > 0.
001B 0360       00234        rlf    indir, F     ;Restore register to its original condition.
001C 0425       00235        bcf    serial,din   ;make sure din is low
001D 0800       00236        retlw  no_err       ;Exit with good status.
                00237
                00238 ;*****************************************************************
                00239 ;*                       DIN8                                    *
                00240 ;*****************************************************************
                00241                            ;Din8 will input 8 bits of data from the
                00242                            ;93CX6.  Before calling this routine,
                00243                            ;the FSR must point to the register
                00244                            ;being used to hold the incomming data.
001E            00245 din8
                00246     IF      wire3
                00247 ;set up the RA1 as a input before proceeding
001E 0C02       00248         movlw   b'00000010'     ;set up porta
001F 0005       00249         tris    serial          ;
                00250     ENDIF
0020 0C08       00251         movlw   8               ;Initialize loop counter.
0021 003F       00252         movwf   cnt             ;
                00253
0022            00254 d_i_8
0022 0360       00255         rlf     indir, F        ;Make room for the incomming bit in the
                00256                                 ;destination register.
0023 0400       00257         bcf     indir,0         ;Assume that the incomming bit is a '0' and
                00258                                 ;clear the LSB of the destination register.
                00259         clkit                   ;Clock a bit in the 93CX6.
                   M                                  ;of the 93CX6 device.
0024 0565          M         bsf     serial,clock    ;Clock (CLK) = '1'.
                   M
0025 0000          M         nop                     ;Adjust the number of nop instructions
                   M                                 ;between the assertion and de-assertion of
                   M                                 ;CLK in proportion to the PIC operating
                   M                                 ;frequency.  Refer to the 93CX6 data for the
                   M                                 ;minimum CLK period.
                   M
0026 0465          M         bcf     serial,clock    ;Clock (CLK) = '0'.
0027 0625       00260        btfsc   serial,dout     ;Test the incomming bit, if our assumption
                00261                                ;was correct, skip the next instruction.
0028 0500       00262        bsf     indir,0         ;No, actual bit is a '1'.  Set the LSB of
                00263                                ;the destination register.
0029 02FF       00264        decfsz  cnt, F          ;Repeat until cnt = 0.
002A 0A22       00265        goto    d_i_8           ;Cnt still > 0.
                00266     IF      wire3
                00267 ;setup RA1 back to output
002B 0C00       00268         movlw   0               ;set RA1 as output
002C 0005       00269         tris    serial          ;      /
                00270     ENDIF
002D 0800       00271         retlw   no_err          ;Exit with good status.
                00272
                00273 ;*****************************************************************
                00274 ;*                      RDYCHK                                   *
                00275 ;*****************************************************************
                00276                           ;Rdychk will read the 93CX6 READY/BUSY status
                00277                           ;and wait for RDY status within the alloted
                00278                           ;number of processor cycles.  If RDY status
```


© 1997 Microchip Technology Inc.

```
                00279                                   ;is not present after this set period, the
                00280                                   ;routine will return with an error status.
                00281
002E            00282 rdychk
                00283     IF      wire3
                00284 ;set up RA1 as a input before proceeding
002E 0C02       00285         movlw   b'00000010'       ;set up porta
002F 0005       00286         tris    serial            ;
                00287     ENDIF
0030 0C20       00288         movlw   tries             ;Initialize time-out counter.
0031 003E       00289         movwf   cnthi             ;
0032 007F       00290         clrf    cnt               ;
                00291         dsel                      ;De-select the 93CX6.
0033 0445           M         bcf     serial,cs         ;Chip Select (CS) = '0' to de-select
                    M                                   ;the device.
                00292
                00293 ;       nop                       ;NOTE:  Check the 93CX6 data sheet for
                00294                                   ;minimum CS low time.  Depending upon
                00295                                   ;processor frequency, a nop(s) may be
                00296                                   ;between the assertion and de-assertion of
                00297                                   ;Chip Select.
                00298
                00299         sel                       ;Re-select the 93CX6.
0034 0545           M         bsf     serial,cs         ;Chip Select (CS) = '1' to select
0035 0625       00300 notrdy  btfsc   serial,dout        ;If DO is a '0', 93CX6 has yet to completed
                00301                                   ;the last operation (still busy).
0036 0A3E       00302         goto    rdynoerr          ;skip to no error
0037 02FF       00303         decfsz  cnt, F            ;No, not yet ready.  Decrement the LSB of
                00304                                   ;our 16 bit timer and check for expiration.
0038 0A35       00305         goto    notrdy            ;Still some time left.  Try again.
0039 02FE       00306         decfsz  cnthi, F          ;Least significant byte expired - decrement
                00307                                   ;and check for expiration of the MSB.
003A 0A35       00308         goto    notrdy            ;Still some time left.  Try again.
                00309     IF      wire3
                00310 ;setup RA1 back to output
003B 0C00       00311         movlw   0                 ;set RA1 as output
003C 0005       00312         tris    serial            ;      /
                00313     ENDIF
003D 0801       00314         retlw   err1              ;RDY status was not present in the alloted
                00315                                   ;time, return with error status.
003E            00316 rdynoerr
                00317     IF      wire3
                00318 ;setup RA1 back to output
003E 0C00       00319         movlw   0                 ;set porta as output
003F 0005       00320         tris    serial            ;      /
                00321     ENDIF
0040 0800       00322         retlw   no_err
                00323
                00324 ;*****************************************************************
                00325 ;*                       SEE                                     *
                00326 ;*****************************************************************
                00327
                00328                               ;See will control the entire operation of a
                00329                               ;93CX6 device.  Prior to calling the routine,
                00330                               ;load a valid command/memory address into
                00331                               ;location cmd, and for WRITE or WRAL
                00332                               ;commands, load registers highb and lowb with
                00333                               ;16 bits of write data.  Upon exit, the W
                00334                               ;register will contain the completion status.
                00335                               ;Only 93CX6 instructions which require a
                00336                               ;status check can return with an error as the
                00337                               ;completion status.  The values that denote
                00338                               ;the completion status are defined as
                00339                               ;variables 'error' and 'no_err' in the
                00340                               ;general assignments section.
                00341
```

7

```
0041                 00342 see
0041 021A            00343         movf    cmd,W           ;save cmd
0042 003E            00344         movwf   temp_cmd
0043 021B            00345         movf    addr,W          ;save addr
0044 003F            00346         movwf   temp_addr       ;
0045 0C1A            00347         movlw   cmd             ;Load W with the location of the cmd
                     00348                                 ;register.
0046 0024            00349         movwf   fsr             ;Transfer that information into the
                     00350                                 ;File Select Register. The fsr now
                     00351                                 ;points to location cmd.
                     00352         sel                     ;Select the 93CX6.
0047 0545               M          bsf     serial,cs       ;Chip Select (CS) = '1' to select
                     00353         strtbt                  ;Send a start bit.
0048 0525               M          bsf     serial,din      ;Start Bit = '1'.
                        M          clkit                   ;Clock it out.
                        M                                  ;of the 93CX6 device.
0049 0565               M          bsf     serial,clock    ;Clock (CLK) = '1'.
                        M
004A 0000               M          nop             ;Adjust the number of nop instructions
                        M                          ;between the assertion and de-assertion of
                        M                          ;CLK in proportion to the PIC operating
                        M                          ;frequency.  Refer to the 93CX6 data for the
                        M                          ;minimum CLK period.
                        M
004B 0465               M          bcf     serial,clock    ;Clock (CLK) = '0'.
                     00354 ;
004C 06FA            00355         btfsc   cmd,7           ;bit 7 = 0?
                     00356      IF      XC46
                     00357         goto    sca8            ;xfer 8 bit cmd/adr
                     00358      ENDIF
                     00359      IF      LC468
                     00360         goto    sca9            ;xfer 9 bit cmd/adr
                     00361      ENDIF
                     00362      IF      H16
004D 0A73            00363         goto    sca10           ;xfer 10 bit cmd/adr
                     00364      ENDIF
                     00365      IF      H8
                     00366         goto    sca11           ;xfer 11 bit cmd/adr
                     00367      ENDIF
                     00368
                     00369 ;         goto    set_cmd_addr   ;no then set cmd/addr
004E 07DA            00370         btfss   cmd,6           ;bit 6 = 0 ?
                     00371      IF      XC46
                     00372         goto    sc8             ;xfer 8 bit cmd/adr
                     00373      ENDIF
                     00374      IF      LC468
                     00375         goto    sc9             ;xfer 9 bit cmd/adr
                     00376      ENDIF
                     00377      IF      H16
004F 0A75            00378         goto    sc10            ;xfer 10 bit cmd/adr
                     00379      ENDIF
                     00380      IF      H8
                     00381         goto    sc11            ;xfer 11 bit cmd/adr
                     00382      ENDIF
                     00383 ;         goto    set_cmd        ;yes then set cmd
                     00384      IF      XC46
                     00385         goto    sca8            ;xfer 8 bit cmd/adr
                     00386      ENDIF
                     00387      IF      LC468
                     00388         goto    sca9            ;xfer 9 bit cmd/adr
                     00389      ENDIF
                     00390      IF      H16
0050 0A73            00391         goto    sca10           ;xfer 10 bit cmd/adr
                     00392      ENDIF
                     00393      IF      H8
                     00394         goto    sca11           ;xfer 11 bit cmd/adr
```


© 1997 Microchip Technology Inc.

```
                00395     ENDIF
                00396
                00397 ;         goto    set_cmd_addr    ;else set cmd/addr
0051            00398 see1
0051 021E       00399           movf    temp_cmd,W      ;retore cmd
0052 003A       00400           movwf   cmd             ;       /
0053 021F       00401           movf    temp_addr,W     ;restore addr
0054 003B       00402           movwf   addr            ;       /
0055 06DA       00403           btfsc   cmd,6           ;Check for a WRITE or ERASE command.
0056 0A5F       00404           goto    see2            ;Yes, parse the command further.
0057 06FA       00405           btfsc   cmd,7           ;Check for a READ command.
0058 0A69       00406           goto    read_           ;Yes, process READ command.
0059 06BA       00407           btfsc   cmd,5           ;Check for a EWEN or ERAL command.
005A 0A66       00408           goto    see3            ;Yes, parse the command further.
005B 069A       00409           btfsc   cmd,4           ;Check for a WRAL command.
005C 0A6E       00410           goto    write_          ;Yes, process WRITE/WRAL command.
                00411
                00412 exit_     dsel                    ;No further processing required; 93CX6
005D 0445          M            bcf     serial,cs       ;Chip Select (CS) = '0' to de-select
                   M                                    ;the device.
                00413                                   ;command completed.
                00414 ;
005E 0800       00415           retlw   no_err          ;Return with good completion status.
                00416
005F 07FA       00417 see2      btfss   cmd,7           ;Check for a ERASE command.
0060 0A6E       00418           goto    write_          ;No, process WRITE command.
0061 092E       00419 exit2_    call    rdychk          ;ERASE command requires a status check.
                00420           dsel                    ;De-select the 93CX6.
0062 0445          M            bcf     serial,cs       ;Chip Select (CS) = '0' to de-select
                   M                                    ;the device.
0063 01E2       00421           addwf   pc, F           ;Compute completion status from
                00422                                   ;results of status check.
0064 0800       00423           retlw   no_err          ;Return with good completion status.
0065 0801       00424           retlw   err1            ;Return with bad completion status.
                00425
0066 069A       00426 see3      btfsc   cmd,4           ;Check for a EWEN command.
0067 0A5D       00427           goto    exit_           ;Yes, no further processing required,
                00428                                   ;exit now.
0068 0A61       00429           goto    exit2_          ;No, ERAL command which requires a
                00430                                   ;status check.
                00431
0069 02A4       00432 read_     incf    fsr, F          ;Increment the File Select Register to
                00433                                   ;point to the register receiving the upper
                00434                                   ;byte of the incomming 93CX6 data word.
006A 091E       00435           call    din8            ;Input the upper byte.
                00436     IF      org8
                00437     ELSE
006B 02A4       00438           incf    fsr, F          ;Increment the File Select Register to point
                00439                                   ;to the register receiving the lower byte.
006C 091E       00440           call    din8            ;Input 8 more bits.
                00441     ENDIF
006D 0A5D       00442           goto    exit_           ;No further processing required, exit now.
                00443
006E 02A4       00444 write_    incf    fsr, F          ;Increment the File Select Register to point
                00445                                   ;to the upper byte of the 16 bit 93CX6 data
                00446                                   ;word to be transmitted.
006F 0910       00447           call    dout8           ;Output that byte.
                00448     IF      org8
                00449     ELSE
0070 02A4       00450           incf    fsr, F          ;Increment the File Select Register to point
                00451                                   ;to the lower byte.
0071 0910       00452           call    dout8           ;Output the lower byte of the 16 bit 93CX6
                00453                                   ;data word.
                00454     ENDIF
0072 0A61       00455           goto    exit2_          ;Exit with a status check.
                00456
```


© 1997 Microchip Technology Inc.

```
                    00457 ;
                    00458 ;
                    00459     IF      XC46
                    00460 sca8
                    00461         movlw   b'11000000'     ;clr all but hi 2
                    00462         andwf   cmd,W           ;save in w
                    00463         iorwf   addr            ;mask in addr.
                    00464         incf    fsr             ;inc FSR
                    00465         call    dout8           ;output
                    00466         goto    see1            ;return
                    00467     ENDIF
                    00468     IF      LC468
                    00469 ;
                    00470 sca9
                    00471         bcf     addr,7          ;clr hi of addr
                    00472         btfsc   cmd,6           ;xfer cmd's bit 6
                    00473         bsf     addr,7          ;to addr's bit 7
                    00474         call    dout9           ;output
                    00475         goto    see1            ;return
                    00476     ENDIF
                    00477     IF      H16
                    00478 ;
0073                00479 sca10
0073 0901           00480         call    dout10          ;output cmd reg
0074 0A51           00481         goto    see1            ;return
                    00482     ENDIF
                    00483 ;
                    00484     IF      H8
                    00485 sca11
                    00486         bcf     cmd,5           ;xfer cmd's bit 3 to
                    00487         btfsc   cmd,3           ;cmd's bit 5
                    00488         bsf     cmd,5           ;       /
                    00489         call    dout11          ;output
                    00490         goto    see1            ;return
                    00491 ;
                    00492     ENDIF
                    00493 ;
                    00494 ;
                    00495 ;
                    00496     IF      XC46
                    00497 sc8
                    00498         movf    cmd,W           ;get command
                    00499         movwf   addr            ;save in addr
                    00500         incf    fsr             ;inc pointer
                    00501         call    dout8           ;output
                    00502         goto    see1            ;return
                    00503 ;
                    00504     ENDIF
                    00505     IF      LC468
                    00506 sc9
                    00507         rlf     cmd,W           ;rotate cmd left
                    00508         movwf   addr            ;save in addr
                    00509         call    dout9           ;xfer 9 bits
                    00510         goto    see1            ;return
                    00511     ENDIF
                    00512     IF      H16
                    00513 ;
0075                00514 sc10
0075 037A           00515         rlf     cmd, F          ;rotate cmd
0076 035A           00516         rlf     cmd,W           ;left twice
0077 003B           00517         movwf   addr            ;save in addr
0078 007A           00518         clrf    cmd             ;clear command
0079 0901           00519         call    dout10          ;xfer 10 bits
007A 0A51           00520         goto    see1            ;return
                    00521     ENDIF
                    00522 ;
```


© 1997 Microchip Technology Inc.

```
                00523     IF      H8
                00524 sc11
                00525         bcf     addr,7          ;xfer cmd's bit 4
                00526         btfsc   cmd,4           ;to addr's bit 7
                00527         bsf     addr,7          ;       /
                00528         call    dout11          ;xfer 11 bits
                00529         goto    see1            ;return
                00530     ENDIF
                00531 ;*****************************************************************
                00532 ;*                      Test Program                             *
                00533 ;*****************************************************************
                00534 ;
007B            00535 main                            ;We've include a sample program to
                00536                                 ;exercise the PIC to 93C66 interface
                00537                                 ;using a simple erase, write and verify
                00538                                 ;routine.8 bit organization has been
                00539                                 ;used with a 3 wire interface.
                00540
007B 0065       00541         clrf    serial          ;Clear the port tied to the 93C66 device.
007C 0CF0       00542         movlw   b'11110000'     ;Intialize the data direction register
007D 0005       00543         tris    serial          ;for that port.
                00544
007E 0C30       00545         movlw   ewen            ;Load W with the Erase/Write Enable command.
007F 003A       00546         movwf   cmd             ;Transfer W into cmd register.
0080 0941       00547         call    see             ;Enable the 93C66 device.
                00548
0081 0C20       00549         movlw   eral            ;Load W with the Erase All command.
0082 003A       00550         movwf   cmd             ;Transfer W into cmd register.
0083 0941       00551         call    see             ;Erase the 93C66.
0084 0F01       00552         xorlw   err1            ;Check completion status.
0085 0643       00553         btfsc   status, zflag   ;Test for error condition.
0086 0AA6       00554         goto    errloop         ;Yes, bad completion status, error-out.
                00555
                00556                                 ;Write loop:
  000000AA      00557 tstptrn equ     0xAA            ;Define the test pattern to be written.
                00558
0087 0C40       00559         movlw   write           ;Load W with the Write command.
0088 003A       00560         movwf   cmd             ;Transfer W into cmd register.
0089 0CAA       00561         movlw   tstptrn         ;Intialize the 93C66 data registers with
                00562                                 ;write data.
008A 003C       00563         movwf   highb           ;load in high byte only
                00564                                 ;since 8 bit low byte is ignored
008B 007B       00565         clrf    addr            ;start at addr 0
008C 0941       00566 test1   call    see             ;Write data word into 93C66 device.
008D 0F01       00567         xorlw   err1            ;Check completion status.
008E 0643       00568         btfsc   status,zflag    ;Test for error condition.
008F 0AA6       00569         goto    errloop         ;Yes, bad completion status, error-out.
0090 03FB       00570         incfsz  addr, F         ;No, increment the 8 bit memory address
                00571                                 ;field.
0091 0A8C       00572         goto    test1           ;write another location
0092 077A       00573         btfss   cmd,3           ;see if all done
0093 0A95       00574         goto    wrt_nxt_pg      ;no then write next page
0094 0A97       00575         goto    read_tst        ;read written data
0095            00576 wrt_nxt_pg
0095 057A       00577         bsf     cmd,3           ;set page bit
0096 0A8C       00578         goto    test1           ;No, write another location.
                00579
                00580                                 ;Read loop:
0097            00581 read_tst
0097 0C80       00582         movlw   read            ;Load W with the Read command.
0098 003A       00583         movwf   cmd             ;Transfer W into cmd register.
0099 0941       00584 test2   call    see             ;Read addressed data word from 93C66 device.
009A 0CAA       00585         movlw   tstptrn         ;Load W with the pattern written.
009B 009C       00586         subwf   highb,W         ;Verify the data read against what was
                00587                                 ;written.
009C 0743       00588         btfss   status,zflag    ;Same?
```

7

```
009D 0AA6       00589          goto    errloop         ;No, error-out.
009E 03FB       00590          incfsz  addr, F         ;Yes, both byte correct, increment the 8 bit
                00591                                  ;memory address field.
009F 0A99       00592          goto    test2           ;do next byte
00A0 077A       00593          btfss   cmd,3           ;check page bit
00A1 0AA3       00594          goto    rd_nxt_pg       ;no then chk next page
00A2 0AA5       00595          goto    allok           ;all done!!!
00A3            00596 rd_nxt_pg
00A3 057A       00597          bsf     cmd,3           ;set page bit
00A4 0A99       00598          goto    test2           ;No, read another location.
                00599
00A5 0AA5       00600 allok    goto    allok           ;Home safe!
                00601 ;
                00602 ;
00A6            00603 errloop
00A6 0AA6       00604          goto    errloop
                00605 ;
                00606 ;
                00607 ;
                00608 ;KEY DEFINITIONS
                00609 ;
01FF            00610          ORG     PIC54
01FF            00611 SYS_RESET
01FF 0A00       00612          GOTO    START
                00613 ;
                00614          END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXX--------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   168
Program Memory Words Free:   344


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: 93C46.ASM

```
MPASM 01.40 Released            93C46.ASM   1-16-1997  13:13:14         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;*******************************************************************
                     00002 ;*                      PICALC Directives Section                  *
                     00003 ;*******************************************************************
                     00004
                     00005         SUBTITL "93C46 3 WIRE INTERFACE ROUTINE"
                     00006         LIST     P=16C54,N=40,C=132
                     00007
                     00008 ;*******************************************************************************
                     00009 ;
                     00010 ;
                     00011 ;       Program:          93C46.ASM
                     00012 ;       Revision Date:
                     00013 ;                         1-13-97      Compatibility with MPASMWIN 1.40
                     00014 ;
                     00015 ;*******************************************************************************
                     00016 ;
                     00017
                     00018 ;*******************************************************************
                     00019 ;*                      Register Assignments                       *
                     00020 ;*******************************************************************
                     00021
00000000             00022 indir   equ     0x00          ;Use this register as source/destination for
                     00023                               ;indirect addressing.
00000002             00024 pc      equ     0x02          ;PIC Program Counter.
00000003             00025 status  equ     0x03          ;PIC Status Register.
00000004             00026 fsr     equ     0x04          ;File Select Register.
00000005             00027 serial  equ     0x05          ;Port used for 93C46 control. Port A is
                     00028                               ;4 bits wide, we'll use 3 or 4 of Port A.
00000001             00029 F       EQU     1
                     00030                               ;The following three registers must be
                     00031                               ;located consecutively in memory.
00000010             00032 cmd     equ     0x10          ;This register contains the 2 bit 93C46
                     00033                               ;command is the upper 2 bit positions and
                     00034                               ;memory address in the lower 6.
00000011             00035 highb   equ     0x11          ;Used in read/write routines to store the
                     00036                               ;upper byte of a 16 bit 93C46 data word.
00000012             00037 lowb    equ     0x12          ;Used in read/write routines to store the
                     00038                               ;lower byte of a 16 bit 93C46 data word.
                     00039
00000013             00040 cnthi   equ     0x13          ;Used as the upper byte of a sixteen bit loop
                     00041                               ;counter in RDYCHK routine.
00000014             00042 cnt     equ     0x14          ;Used as the lower byte of a sixteen bit loop
                     00043                               ;counter in RDYCHK routine, and elsewhere as
                     00044                               ;an eight bit counter.
                     00045
                     00046 ;*******************************************************************
                     00047 ;* Bit Assignments: The following assignment is for 3-wire setup.
                     00048 ;* For 4-wire setup assign din and dout to seperate pins.
                     00049
00000000             00050 carry   equ     0             ;Carry Flag of Status Register.
00000002             00051 zflag   equ     2             ;Zero Flag of Status Register.
                     00052
                     00053 ;For the 3 wire interface, connect the din and dout to the same
                     00054 ;i/o line of the PIC16C5X.
```

7

```
00000000      00055 cs      equ     0             ;Port pin tied to CS on 93C46.
00000001      00056 din     equ     1             ;Port pin tied to DI on 93C46. 3-wire
00000001      00057 dout    equ     1             ;Port pin tied to DO on 93C46. 3-wire
00000002      00058 clock   equ     2             ;Port pin tied to CLK on 93C46.
                    00059
                    00060 ;*****************************************************************
                    00061 ;*                       General Assignments                     *
                    00062 ;*****************************************************************
                    00063
00000000      00064 no_err  equ     0             ;
00000001      00065 err1    equ     1             ;
00000004      00066 tries   equ     0x04          ;After issuing a WRITE, ERASE, ERAL, or WRAL
                    00067                       ;command, the approximate number of machine
                    00068                       ;cycles X 256 to wait for the RDY status.
                    00069                       ;This value must be adjusted for operating
                    00070                       ;frequencies other than 4 MHz.
                    00071
00000080      00072 read    equ     0x80          ;93C46 Read command.
00000040      00073 write   equ     0x40          ;93C46 Write command.
000000C0      00074 erase   equ     0xC0          ;93C46 Erase command.
00000030      00075 ewen    equ     0x30          ;93C46 Erase/Write Enable command.
00000000      00076 ewds    equ     0x00          ;93C46 Erase/Write Disable command.
00000020      00077 eral    equ     0x20          ;92CXX Erase All command.
00000010      00078 wral    equ     0x10          ;92CXX Write All command.
                    00079
                    00080 ;*****************************************************************
                    00081 ;*                       Macro Definitions                       *
                    00082 ;*****************************************************************
                    00083
                    00084 sel     MACRO                 ;Selects the 93C46 device.
                    00085         bsf     serial,cs     ;Chip Select (CS) = '1' to select the device.
                    00086         ENDM
                    00087
                    00088 dsel    MACRO                 ;De-select the 93C46 device.
                    00089         bcf     serial,cs     ;Chip Select (CS) = '0' to de-select the
                    00090                               ;device.
                    00091         ENDM
                    00092
                    00093 strtbt  MACRO                 ;Issue the Start Bit to the 93C46.
                    00094         bsf     serial,din    ;Start Bit = '1'.
                    00095         clkit                 ;Clock it out.
                    00096         ENDM
                    00097
                    00098 clkit   MACRO                 ;Clocks a serial data bit into or out of the
                    00099                               ;93C46 device.
                    00100         bsf     serial,clock  ;Clock (CLK) = '1'.
                    00101
                    00102         nop                   ;Adjust the number of nop instructions
                    00103                               ;between the assertion and de-assertion of
                    00104                               ;CLK in proportion to the PIC operating
                    00105                               ;frequency.  Refer to the 93C46 data for the
                    00106                               ;minimum CLK period.
                    00107
                    00108         bcf     serial,clock  ;Clock (CLK) = '0'.
                    00109         ENDM
                    00110
                    00111 ;*****************************************************************
                    00112 ;*                    Power-On/Reset Entry Point                 *
                    00113 ;*****************************************************************
                    00114
01FF                00115 reset_  org     0x1FF
01FF 0A56           00116         goto    main
                    00117
                    00118 ;*****************************************************************
                    00119 ;*                       93C46 Routines                          *
                    00120 ;*****************************************************************
```


© 1997 Microchip Technology Inc.

```
0000            00121          org     0x000          ;Locate all subroutines in the lower half of
                00122                                 ;a Program Memory Page.
                00123
                00124 ;*****************************************************************
                00125 ;*                          DOUT8                                *
                00126 ;*****************************************************************
                00127                                 ;Dout8 will output 8 bits of data to the
                00128                                 ;93C46.  Before calling this routine, the FSR
                00129                                 ;must point to the byte being transmitted.
                00130
0000 0C08       00131 dout8    movlw   0x08           ;Initialize loop counter.
0001 0034       00132          movwf   cnt            ;
                00133
0002 0425       00134 d_o_8    bcf     serial,din     ;Assume that the bit to be transfered is a
                00135                                 ;'0'.  Hence, de-assert DI.
0003 0360       00136          rlf     indir, F       ;Rotate the actual bit to be transferred into
                00137                                 ;the carry bit.
0004 0603       00138          btfsc   status,carry   ;Test the carry, if our assumption was
                00139                                 ;correct, skip the next instruction.
0005 0525       00140          bsf     serial,din     ;No, actual bit was a '1'.  Assert DI.
                00141          clkit                  ;Clock the 93C46.
                   M                                  ;93C46 device.
0006 0545          M          bsf     serial,clock   ;Clock (CLK) = '1'.
                   M
0007 0000          M          nop                    ;Adjust the number of nop instructions
                   M                                 ;between the assertion and de-assertion of
                   M                                 ;CLK in proportion to the PIC operating
                   M                                 ;frequency.  Refer to the 93C46 data for the
                   M                                 ;minimum CLK period.
                   M
0008 0445          M          bcf     serial,clock   ;Clock (CLK) = '0'.
0009 02F4       00142          decfsz  cnt, F         ;Repeat until cnt = 0.
000A 0A02       00143          goto    d_o_8          ;Cnt still > 0.
000B 0360       00144          rlf     indir, F       ;Restore register to its original condition.
000C 0800       00145          retlw   no_err         ;Exit with good status.
                00146
                00147 ;*****************************************************************
                00148 ;*                           DIN8                                *
                00149 ;*****************************************************************
                00150                                 ;Din8 will input 8 bits of data from the
                00151                                 ;93C46.  Before calling this routine, the FSR
                00152                                 ;must point to the register being used to
                00153                                 ;hold the incomming data.
000D 0C08       00154 din8     movlw   0x08           ;Initialize loop counter.
000E 0034       00155          movwf   cnt            ;
                00156 ;for the 3 wire interface the direction of the i/o line connected to
                00157 ;din and dout has to converted from an output to an input.
000F 0C02       00158          movlw   b'00000010'    ;convert RA1 to an input
0010 0005       00159          tris    serial         ;       /
                00160
                00161 d_i_8    clkit                  ;Clock a bit out of the 93C46.
                   M                                  ;93C46 device.
0011 0545          M          bsf     serial,clock   ;Clock (CLK) = '1'.
                   M
0012 0000          M          nop                    ;Adjust the number of nop instructions
                   M                                 ;between the assertion and de-assertion of
                   M                                 ;CLK in proportion to the PIC operating
                   M                                 ;frequency.  Refer to the 93C46 data for the
                   M                                 ;minimum CLK period.
                   M
0013 0445          M          bcf     serial,clock   ;Clock (CLK) = '0'.
0014 0360       00162          rlf     indir, F       ;Make room for the incomming bit in the
                00163                                 ;destination register.
0015 0400       00164          bcf     indir,0        ;Assume that the incomming bit is a '0' and
                00165                                 ;clear the LSB of the destination register.
0016 0625       00166          btfsc   serial,dout    ;Test the incomming bit, if our assumption
```


© 1997 Microchip Technology Inc.

```
                00167                                   ;was correct, skip the next instruction.
0017 0500       00168           bsf     indir,0         ;No, actual bit is a '1'.  Set the LSB of the
                00169                                   ;destination register.
0018 02F4       00170           decfsz  cnt, F          ;Repeat until cnt = 0.
0019 0A11       00171           goto    d_i_8           ;Cnt still > 0
                00172 ;for a 3 wire interface, convert the RA1 line back to an output
001A 0C00       00173           movlw   0               ;make RA1 to an output
001B 0005       00174           tris    serial          ;       /
001C 0800       00175           retlw   no_err          ;Exit with good status.
                00176
                00177 ;*****************************************************************
                00178 ;*                       RDYCHK                                  *
                00179 ;*****************************************************************
                00180                                   ;Rdychk will read the 93C46 READY/BUSY status
                00181                                   ;and wait for RDY status within the alloted
                00182                                   ;number of processor cycles.  If RDY status
                00183                                   ;is not present after this set period, the
                00184                                   ;routine will return with an error status.
                00185
001D 0C04       00186 rdychk    movlw   tries           ;Initialize time-out counter.
001E 0033       00187           movwf   cnthi           ;
001F 0074       00188           clrf    cnt             ;
                00189 ;for a 3 wire interface, make the RA1 line an input
0020 0C02       00190           movlw   b'00000010'     ;
0021 0005       00191           tris    serial
                00192           dsel                    ;De-select the 93C46.
0022 0405           M           bcf     serial,cs       ;Chip Select (CS) = '0' to de-select the
                    M                                   ;device.
                00193
                00194 ;         nop                     ;NOTE:  Check the 93C46 data sheet for
                00195                                   ;minimum CS low time.  Depending upon
                00196                                   ;processor frequency, a nop(s) may be
                00197                                   ;between the assertion and de-assertion of
                00198                                   ;Chip Select.
                00199
                00200           sel                     ;Re-select the 93C46.
0023 0505           M           bsf     serial,cs       ;Chip Select (CS) = '1' to select the device.
0024 0625       00201 notrdy    btfsc   serial,dout     ;If DO is a '0', 93C46 has yet to completed
                00202                                   ;the last operation (still busy).
0025 0A2D       00203           goto    no_error        ;Otherwise RDY status is present within the
                00204                                   ;alloted time, and return with good status.
0026 02F4       00205           decfsz  cnt, F          ;No, not yet ready.  Decrement the LSB of our
                00206                                   ;16 bit timer and check for expiration.
0027 0A24       00207           goto    notrdy          ;Still some time left.  Try again.
0028 02F3       00208           decfsz  cnthi, F        ;Least significant byte expired - decrement
                00209                                   ;and check for expiration of the MSB.
0029 0A24       00210           goto    notrdy          ;Still some time left.  Try again.
                00211 ;for a 3 wire interface, convert RA1 line back to an ouput
002A 0C00       00212           movlw   0               ;convert RA1 to an output
002B 0005       00213           tris    serial          ;       /
002C 0801       00214           retlw   err1            ;RDY status was not present in the alloted
                00215                                   ;time, return with error status.
002D            00216 no_error
                00217 ;for a 3 wire interface, convert RA1 line back to an ouput
002D 0C00       00218           movlw   0               ;convert RA1 to an output
002E 0005       00219           tris    serial          ;       /
002F 0800       00220           retlw   no_err
                00221
                00222
                00223 ;*****************************************************************
                00224 ;*                       SEE                                     *
                00225 ;*****************************************************************
                00226
                00227                                   ;See will control the entire operation of a
                00228                                   ;93C46 device.  Prior to calling the routine,
                00229                                   ;load a valid command/memory address into
```


© 1997 Microchip Technology Inc.

```
                    00230                           ;location cmd, and for WRITE or WRAL
                    00231                           ;commands, load registers highb and lowb with
                    00232                           ;16 bits of write data.  Upon exit, the W
                    00233                           ;register will contain the completion status.
                    00234                           ;Only 93C46 instructions which require a
                    00235                           ;status check can return with an error as the
                    00236                           ;completion status.  The values that denote
                    00237                           ;the completion status are defined as
                    00238                           ;variables 'error' and 'no_err' in the
                    00239                           ;general assignments section.
                    00240
0030 0C10           00241 see       movlw   cmd     ;Load W with the location of the cmd
                    00242                           ;register.
0031 0024           00243           movwf   fsr     ;Transfer that information into the File
                    00244                           ;Select Register.  The fsr now points to
                    00245                           ;location cmd.
                    00246           sel             ;Select the 93C46.
0032 0505               M           bsf     serial,cs ;Chip Select (CS) = '1' to select the device.
                    00247           strtbt          ;Send a start bit.
0033 0525               M           bsf     serial,din ;Start Bit = '1'.
                        M           clkit           ;Clock it out.
                        M                           ;93C46 device.
0034 0545               M           bsf     serial,clock ;Clock (CLK) = '1'.
                        M
0035 0000               M           nop             ;Adjust the number of nop instructions
                        M                           ;between the assertion and de-assertion of
                        M                           ;CLK in proportion to the PIC operating
                        M                           ;frequency.  Refer to the 93C46 data for the
                        M                           ;minimum CLK period.
                        M
0036 0445               M           bcf     serial,clock ;Clock (CLK) = '0'.
0037 0900           00248           call dout8      ;Transmit the 2 bit command and six bit
                    00249                           ;address.
0038 06D0           00250           btfsc   cmd,6   ;Check for a WRITE or ERASE command.
0039 0A42           00251           goto    see2    ;Yes, parse the command further.
003A 06F0           00252           btfsc   cmd,7   ;Check for a READ command.
003B 0A4C           00253           goto    read_   ;Yes, process READ command.
003C 06B0           00254           btfsc   cmd,5   ;Check for a EWEN or ERAL command.
003D 0A49           00255           goto    see3    ;Yes, parse the command further.
003E 0690           00256           btfsc   cmd,4   ;Check for a WRAL command.
003F 0A51           00257           goto    write_  ;Yes, process WRITE/WRAL command.
                    00258
                    00259 exit_     dsel            ;No further processing required; 93C46
0040 0405               M           bcf     serial,cs ;Chip Select (CS) = '0' to de-select the
                        M                           ;device.
                    00260                           ;command completed.
0041 0800           00261           retlw   no_err  ;Return with good completion status.
                    00262
0042 07F0           00263 see2      btfss   cmd,7    ;Check for a ERASE command.
0043 0A51           00264           goto    write_  ;No, process WRITE command.
0044 091D           00265 exit2_    call    rdychk  ;ERASE command requires a status check.
                    00266           dsel            ;De-select the 93C46.
0045 0405               M           bcf     serial,cs ;Chip Select (CS) = '0' to de-select the
                        M                           ;device.
0046 01E2           00267           addwf   pc, F   ;Compute completion status from results of
                    00268                           ;status check.
0047 0800           00269           retlw   no_err  ;Return with good completion status.
0048 0801           00270           retlw   err1    ;Return with bad completion status.
                    00271
0049 0690           00272 see3      btfsc   cmd,4   ;Check for a EWEN command.
004A 0A40           00273           goto    exit_   ;Yes, no further processing required, exit
                    00274                           ;now.
004B 0A44           00275           goto    exit2_  ;No, ERAL command which requires a status
                    00276                           ;check.
                    00277
004C 02A4           00278 read_     incf    fsr, F  ;Increment the File Select Register to point
```


© 1997 Microchip Technology Inc.

```
                00279                               ;to the register receiving the upper byte of
                00280                               ;the incomming 93C46 data word.
004D 090D       00281           call    din8        ;Input the upper byte.
004E 02A4       00282           incf    fsr, F      ;Increment the File Select Register to point
                00283                               ;to the register receiving the lower byte.
004F 090D       00284           call    din8        ;Input 8 more bits.
0050 0A40       00285           goto    exit_       ;No further processing required, exit now.
                00286
0051 02A4       00287 write_    incf    fsr, F      ;Increment the File Select Register to point
                00288                               ;to the upper byte of the 16 bit 93C46 data
                00289                               ;word to be transmitted.
0052 0900       00290           call    dout8       ;Output that byte.
0053 02A4       00291           incf    fsr, F      ;Increment the File Select Register to point
                00292                               ;to the lower byte.
0054 0900       00293           call    dout8       ;Output the lower byte of the 16 bit 93C46
                00294                               ;data word.
0055 0A44       00295           goto    exit2_      ;Exit with a status check.
                00296
                00297 ;*****************************************************************
                00298 ;*                       Test Program                            *
                00299 ;*****************************************************************
                00300 ;
0056            00301 main                          ;We've include a sample program to exercise
                00302                               ;the PIC to 93C46 interface using a simple
                00303                               ;erase, write and varify routine.
                00304
0056 0065       00305           clrf    serial      ;Clear the port tied to the 93C46 device.
0057 0CF4       00306           movlw   b'11110100' ;Intialize the data direction register for
0058 0005       00307           tris    serial      ;that port.
                00308
0059 0C30       00309           movlw   ewen        ;Load W with the Erase/Write Enable command.
005A 0030       00310           movwf   cmd         ;Transfer W into cmd register.
005B 0930       00311           call    see         ;Enable the 93C46 device.
                00312
005C 0C20       00313           movlw   eral        ;Load W with the Erase All command.
005D 0030       00314           movwf   cmd         ;Transfer W into cmd register.
005E 0930       00315           call    see         ;Erase the 93C46.
005F 0F01       00316           xorlw   err1        ;Check completion status.
0060 0643       00317           btfsc   status, zflag ;Test for error condition.
0061 0A81       00318           goto    errloop     ;Yes, bad completion status, error-out.
                00319
                00320                               ;Write loop:
  0000001F      00321 loopcnt equ     0x1F          ;Define an unused location for our test
                00322                               ;program loop counter.
  000000AA      00323 tstptrn equ     0xAA          ;Define the test pattern to be written.
                00324
0062 0C40       00325           movlw   .64         ;Initialize that counter.
0063 003F       00326           movwf   loopcnt     ;
0064 0C40       00327           movlw   write       ;Load W with the Write command.
0065 0030       00328           movwf   cmd         ;Transfer W into cmd register.
0066 0CAA       00329           movlw   tstptrn     ;Intialize the 93C46 data registers with
                00330                               ;write data.
0067 0031       00331           movwf   highb       ;
0068 0032       00332           movwf   lowb        ;
0069 0930       00333 test1     call    see         ;Write data word into 93C46 device.
006A 0F01       00334           xorlw   err1        ;Check completion status.
006B 0643       00335           btfsc   status,zflag ;Test for error condition.
006C 0A81       00336           goto    errloop     ;Yes, bad completion status, error-out.
006D 02B0       00337           incf    cmd, F      ;No, increment the 6 bit memory address
                00338                               ;field.
006E 02FF       00339           decfsz  loopcnt, F  ;Have we written all 64 locations?
006F 0A69       00340           goto    test1       ;No, write another location.
                00341
                00342                               ;Read loop:
                00343
0070 0C40       00344           movlw   .64         ;Initialize loop counter.
```


© 1997 Microchip Technology Inc.

```
0071 003F       00345          movwf   loopcnt         ;
0072 0C80       00346          movlw   read            ;Load W with the Read command.
0073 0030       00347          movwf   cmd             ;Transfer W into cmd register.
0074 0930       00348 test2    call    see             ;Read addressed data word from 93C46 device.
0075 0CAA       00349          movlw   tstptrn         ;Load W with the pattern written.
0076 0091       00350          subwf   highb,0         ;Verify the data read against what was
                00351                                  ;written.
0077 0743       00352          btfss   status,zflag    ;Same?
0078 0A81       00353          goto    errloop         ;No, error-out.
0079 0CAA       00354          movlw   tstptrn         ;Repeat with the lower byte read.
007A 0092       00355          subwf   lowb,0          ;
007B 0743       00356          btfss   status,zflag    ;Same?
007C 0A81       00357          goto    errloop         ;No, error-out.
007D 02B0       00358          incf    cmd, F          ;Yes, both byte correct, increment the 6 bit
                00359                                  ;memory address field.
007E 02FF       00360          decfsz  loopcnt, F      ;Have we read all 64 locations?
007F 0A74       00361          goto    test2           ;No, read another location.
                00362
0080 0A80       00363 allok    goto    allok           ;Home safe!
                00364
0081 0A81       00365 errloop  goto    errloop         ;Bad news!
                00366
                00367          END                     ;Thats all folks!
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XX-------------- ---------------- ---------------- ----------------
93C46 3 WIRE INTERFACE ROUTINE
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)


01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   131
Program Memory Words Free:   381


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

7

NOTES:

© 1997 Microchip Technology Inc.

Logic Powered Serial EEPROMs

Authore: *Richard J. Fisher*
Microchip Technology Inc.
Bruce Negley
Microchip Technolgy Inc.

Embedded applications increasingly want more integration and power, in less space for less cost. Using low power Serial EEPROMs (SEE) for application firmware, lookup tables, and microcode coupled with small footprints makes for permanent storage at respectable savings. One additional method of saving on the power budget is selectively powering off components when not needed, a basic for embedded power management. The low-power SEEs offered by Microchip Technology Inc., offer an additional benefit, powering the SEE from a microcontroller port. This allows the host controller to not only manipulate the Serial EEPROM Reads and Write, but also the periods when it is powered off or on. Satellite communications use this technique to save power and total dose accumulation. We call this technique POWER PORT™. The microcontroller port must have sufficient Ioh (source current) to sustain the voltage and current for all memory functions, READ, ERASE, and WRITE. Obviously, not all memory or peripheral devices could be powered thusly, but Microchip's SEE devices will function in this environment.

The microcontroller, using its internal software and hardware decision functions, determines when it needs to communicate with the memory device, then acts accordingly. Any standard wake-up sequence will accomplish this task. The wake-up code needs only power up the memory and wait for the power to become stable before doing a read or write by driving the POWER PORT high. Then all serial communication executes normally. The SEEs are powered off for additional power savings and the data or code is utilized from RAM. Obviously, the port output must be allowed to settle, but normal operation of the output structures would guarantee that this would be met. The I/O port Tpd for the Microchip PIC16C5X, is specified at 40ns maximum.

The 24LCXX and 93LCXX CMOS SEE series parts from Microchip were designed to achieve low current consumption across all ranges of operation.

The four primary ICC parameters for these products are:

Parameter	Conditions
ICC STANDBY	Not in an active operation while VCC is supplied.
ICC READ	The part is in a READ operation.
ICC PEAK WRITE	The BYTE / PAGE WRITE and ERASE operations have self timed cycles of 10 ms. A typical of 4 ms is the actual time of the operation. This is the amount of time when the ICC requires the most current (PEAK WRITE). The part is drawing STANDBY ICC during the remaining 6ms of the cycle.
ICC AVG WRITE	The avg of the PEAK WRITE ICC and STANDBY ICC during the self-timed 10ms write cycle.

The attached characteristic curves (Figure 1 and Figure 2) indicate that ICC PEAK WRITE current consumes the most current. The worst case condition is at 6.0V and –40°C. The 24LCXX series parts draw a typical 3.2 mA and the 93LCXX series parts draw a typical of 2.0 mA. These low ICC characteristics offer a unique current saving benefit for battery applications. Figure 3 and Figure 4 illustrate the sink and source current capabilities of the PIC16C5X family of microcontrollers. It is clear from these characterization curves that the microcontroller can deliver sufficient current across all temperature ranges to power a SEE using the POWER PORT technique.

Figure 5 shows the connection scheme for the Microchip PIC16C54. It should be noted that not all versions of competitive microcontrollers are capable of powering a device in this manner and the specific data sheets for the microcontroller being considered must be consulted for maximum source current. The microcontroller port must be capable of sourcing sufficient current for the duration of the write cycle or 10ms, worse case. The peak write requirement for the 24LCXX product family is 3.2 mA at 5.5 Vdc (–40°C).

Listing A demonstrates the appropriate code sequences when using the PIC16C54 microcontroller. The sequences included are power control, start bit, stop bit, send and receive bit, Tx and Rx, and a general addressing routine.

© 1997 Microchip Technology Inc.

FIGURE 1: TYPICAL Icc FOR 24LCXX

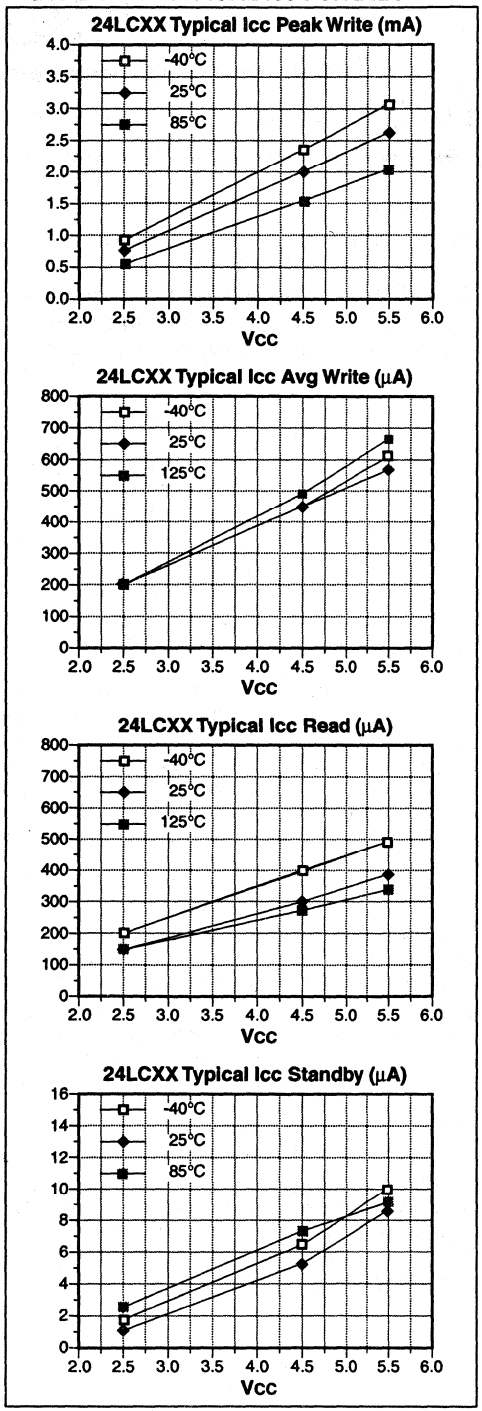
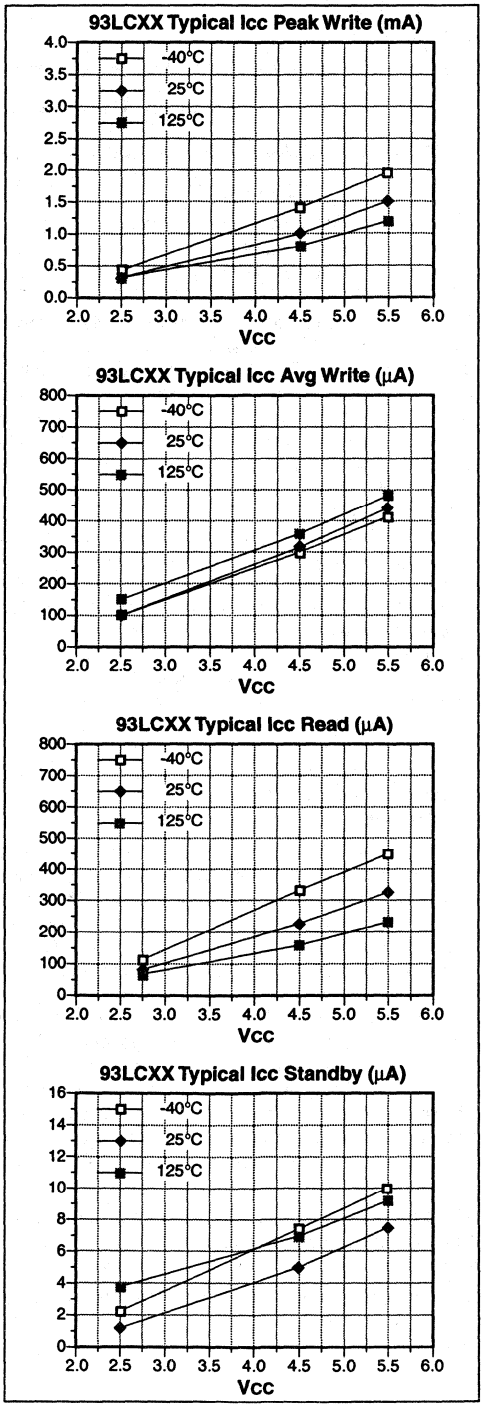


FIGURE 2: TYPICAL Icc FOR 93CXX



© 1997 Microchip Technology Inc.

FIGURE 3: PIC16C5X I_{OL} AT 5V

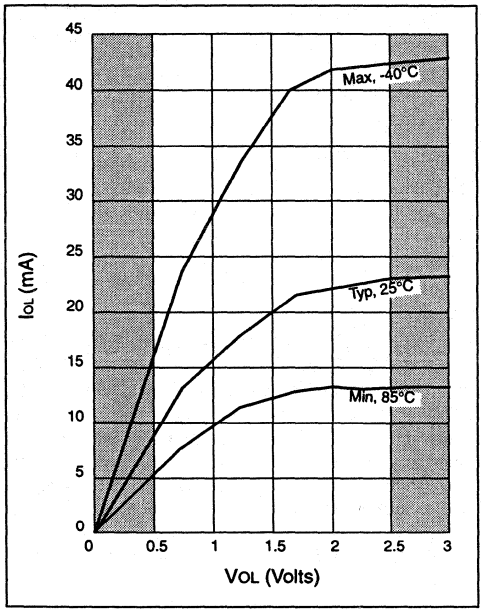


FIGURE 4: PIC16C5X I_{OH} AT 5V

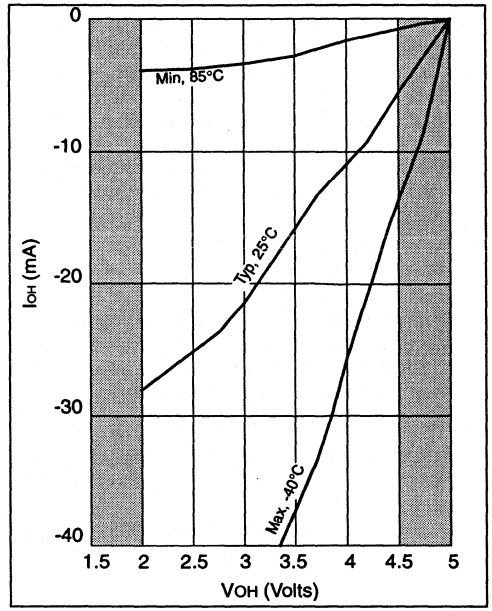
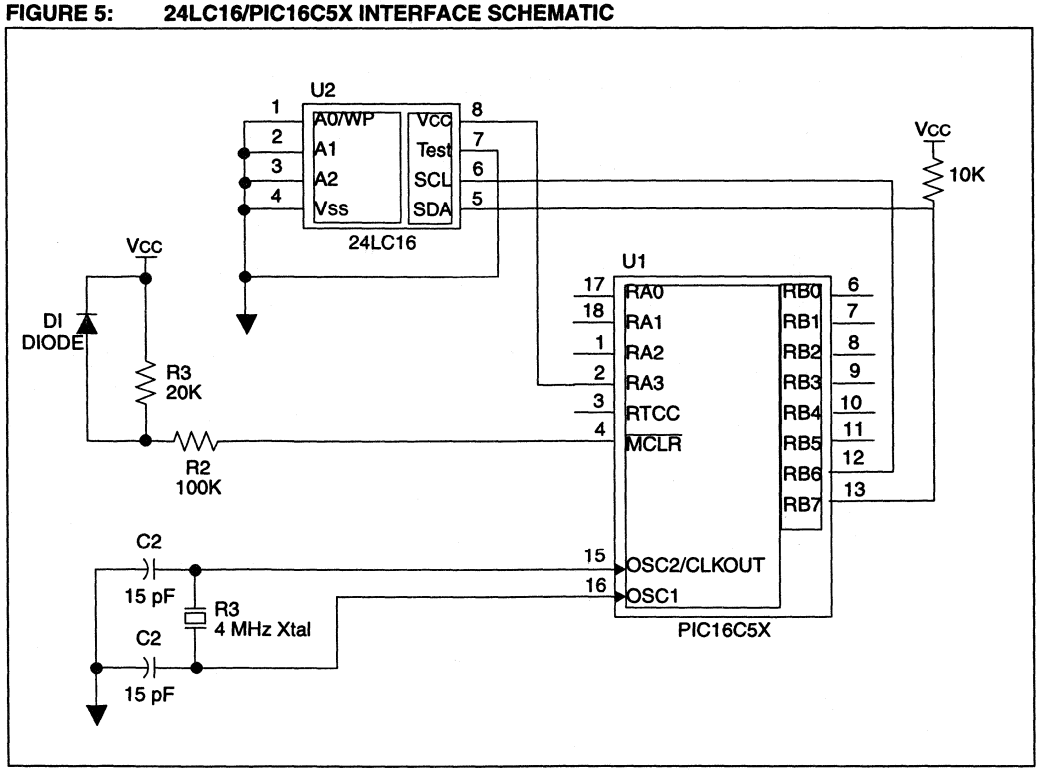


FIGURE 5: 24LC16/PIC16C5X INTERFACE SCHEMATIC



7

The primary benefits of this application are:

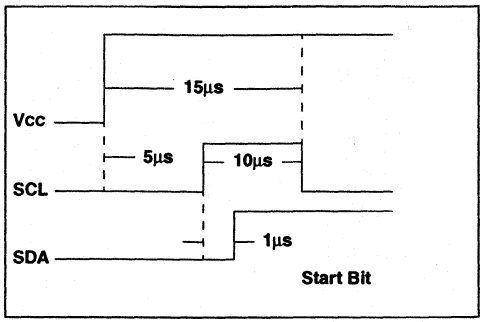
- The SEE is completely powered down to save power when the SEE is not executing an operation. This will directly effect the total system power consumption. This means that the SEE is in a total quiescent state and even the standby current savings are realized, greatly increasing usable battery life, and consequently allowing for a more sophisticated design on the same power budget.
- The very fast 5 µs power-up time minimizes power-up delay.
- Since the serial operation is gated by a stable microcontroller V_{OH}, risk of data being corrupted by a glitch is minimized. This, in effect, is a regulated V_{CC} supply and provides a reliable power source to ensure data integrity.

Several cautions need to be noted:

1. *Gang powering multiple devices must not exceed the I/O port I_{OH} or capacitive load specifications.*
2. *The total power requirements vs. power budget must be considered, including the extra drain on the microcontroller.*
3. *The microcontroller I_{CC} max must not be exceeded.*
4. *Normal decoupling methods must be employed.*
5. *The microcontroller I_{OH} for the port in use must not be exceeded.*

Figure 6 shows a typical power on to start bit sequence. Notice that the device is available to receive a clock at 5 µs after V_{CC} has become stable.

FIGURE 6:



Many applications, especially remote or handheld data acquisition applications, where power consumption is at a premium or battery life is critical can use the POWER PORT technique with the PICmicro™ microcontrollers and possibly other microcontrollers. Remote metering applications where the microcontroller must wake up and report previously stored data or periodically sample inputs, such as gas, electrical, or water monitoring systems are good examples where POWER PORT would be beneficial. Underground monitoring equipment for fuel storage and environmental monitoring systems are also suitable applications.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
LIST P=16C54
;
;
;   Sample test program to power up serial EEPROM
;   using PIC16/17 port A, then write one byte and read same byte, then repeat forever.
;
;*****************************************************************************************
port_a  equ     5h      ; port 5 used for device
                        ; address select
port_b  equ     6h      ; port 6 used for data and
                        ; clock lines
eeprom  equ     0ah     ; bit buffer
addr    equ     0ch     ; address register
datai   equ     0dh     ; stored data input reg.
datao   equ     0eh     ; stored data output reg.
slave   equ     0fh     ; device address
                        ; (1010xxx0)
txbuf   equ     10h     ; tx buffer
count   equ     11h     ; bit counter
bcount  equ     12h     ; byte counter
rxbuf   equ     13h     ; receive buffer
loops   equ     15h     ; delay loop counter
loops2  equ     16h     ; delay loop counter 2
;
;       Bit Assignments
;
di      equ     7       ; eeprom input
do      equ     6       ; eeprom output
sdata   equ     7       ; data line (port_b)
sclk    equ     6       ; clock line (port_b)
vcc     equ     3       ; vcc for dut (port_a)
;
        org     01ffh
begin   goto    PWRUP
        org     000h
        goto    PWRUP
;
;*****************************************************************************************
;        DELAY ROUTINE
;        this routine takes the value in loops and loops that many times.  Every
;        increase in 'loops' yields approx 1 more millisecond.
;        i.e., if 'loops' is 10 then the wait period is approx 10 milliseconds.
;
;-----------------------------------------------------------------------------------------
WAIT
;

top2    movlw   .110
        movwf   loops2
top     nop                     ; sit and wait
        nop
        nop
        nop
        nop
        nop
        decfsz  loops2; inner loop done?
        goto    top; no, go again
        decfsz  loops   ; outer loop done?
        goto    top2 ; no, go again
        retlw   0; yes, return from sub
```


© 1997 Microchip Technology Inc.

```
;
;*****************************************************************************************
;       Start Bit Subroutine
;       this routine generates a start bit
;-----------------------------------------------------------------------------------------
;
BSTART
        movlw   b'00111111'
        tris    port_b          ; port b for output
        bsf     port_b,sdata    ; set clock high
        nop
        nop
        bsf     port_b,sclk     ; set clock high
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        bcf     port_b,sdata    ; data line goes low during high clock for start bit
        nop
        nop
        nop
        nop
        nop
        nop
        bcf     port_b,sclk     ; start clock train
        nop
        nop
        nop
        retlw   0
         ;
         ;    End of Subroutine
;*****************************************************************************************
;
;        Stop Bit Subroutine
;        this routine generates a stop bit
;-----------------------------------------------------------------------------------------
BSTOP
        movlw   b'00111111'     ;
        tris    port_b          ; set data/clock lines as outputs
        bcf     port_b,sdata    ; make sure data line is low
        nop
        nop
        nop
        nop
        nop
        nop
        bsf     port_b,sclk     ; set clock high
        nop
        nop
        nop
        nop
        nop
        nop
        bsf     port_b,sdata    ; data goes high while clock high
                                ; for stop bit
        nop
        nop
        nop
        nop
        nop
        bcf     port_b,sclk     ; set clock low again
        nop
```


© 1997 Microchip Technology Inc.

```
        nop
        nop
        retlw   0
;
;           End of Subroutine
;*****************************************************************************
;       Serial data send 1 bit from PIC16/17 to dut
;-----------------------------------------------------------------------------
BITOUT
        movlw   b'00111111'     ; set data,clock as outputs
        tris    port_b
        btfss   eeprom,do
        goto    BIT0
        bsf     port_b,sdata    ; output bit 0
        goto    CLK1            ; data line clocked low by device
;
BIT0
        bcf     port_b,sdata    ; output bit 0
CLK1
        nop
        nop
        bsf     port_b,sclk     ; set clock line high
BIT2
        nop
        nop
        nop
        nop
        bcf     port_b,sclk     ; return clock line low
        retlw   0
;
;       End of Subroutine
;
;*****************************************************************************************
;    Bit in routine
;     this routine gets a bit of data from the part
;               into the 'eeprom' register, bit 'di'
;-----------------------------------------------------------------------------
BITIN
        movlw   b'10111111'     ; make sdata an input line
        tris    port_b
        bcf     eeprom,di       ; assume input bit low
        bsf     port_b,sclk     ; set clock line high
        nop                     ; just sit here a sec
        nop
        nop
        nop
        nop
        nop
        nop
        nop                     ;
        btfsc   port_b,sdata    ; read data line
        bsf     eeprom,di       ; set input bit if needed
        bcf     port_b,sclk     ; set clock line low
        retlw   0               ; hit the road
;
;*****************************************************************************************
;
;    Transmit Data Subroutine
;-----------------------------------------------------------------------------
TX
        movlw   .8
        movwf   count           ; set the #bits to 8
        ;
TXLP
        bcf     eeprom,do
        btfsc   txbuf,7
```


© 1997 Microchip Technology Inc.

```
        bsf     eeprom,do       ; otherwise data bit =1
        call    BITOUT          ; serial data out
        rlf     txbuf           ; rotate txbuf left
        decfsz  count           ; 8 bits done?
        goto    TXLP      ; no - go again
        call    BITIN           ; read ack bit
        ;
        retlw   0
;       End of Subroutine
;*****************************************************************************************
;        Receive data Routine
;        this routine gets a byte of data from the part into 'rxbuf'
;-----------------------------------------------------------------------------------------
RX
        movlw   .8              ; set # bits to 8
        movwf   count
        clrf    rxbuf           ; clear receive buffer
RXLP    rlf     rxbuf           ; rotate buffer left 1 bit
        bcf     rxbuf,0         ; assume bit is zero
        call    BITIN           ; read a bit
        btfsc   eeprom,di       ; input bit high?
        bsf     rxbuf,0         ; yes, set buffer bit high
        decfsz  count           ; 8 bits done?
        goto    RXLP            ; no, do another
        bcf     eeprom,do       ; set ack bit = 0
        call    BITOUT          ; to finish transmission
        retlw   0
;
;*****************************************************************************************
;        Power up routine
;    this routine blinks the lights
;-----------------------------------------------------------------------------------------
PWRUP
        movlw  b'00000001'
        tris   port_a           ; set RA0 as input, rest output
        bsf    port_a,vcc       ; turn on power to dut
        nop                     ; wait for dut to power up
        nop
        nop
        nop
        nop
;
;*****************************************************************************************
;         Byte Write Routine
;         this writes the data in "55h" to the first byte
;         in the serial EEPROM.
;-----------------------------------------------------------------------------------------
;
WRBYTE
                        ;
        movlw   b'10100000' ; set slave address and write mode
        movwf  slave
        movlw   b'01010101' ; set data to 55h
        movwf datao
                        ;
        clrf    addr        ; set address to 00h
                        ;
        call    BSTART      ; generate start bit
        movf    slave,w     ; get slave address
        movwf   txbuf       ; into transmit buffer
        call    TX          ; and send it
        movf    addr,w      ; get word address
        movwf   txbuf       ; into transmit buffer
        call    TX          ; and send it
        movf    datao,w     ; move data
        movwf   txbuf       ; to tranmit buffer
```


© 1997 Microchip Technology Inc.

```
        call    TX              ; and transmit it
        call    BSTOP           ; generate stop bit
                                ;
        movlw                  .10
        movwf   loops           ; set delay time to give
        call    WAIT            ; 10 ms wait after every byte
                                ;
                                ; now drop through and do the read
                                ;
;*****************************************************************************************
;         READ (read routine)
;         this routine reads the first address
;         of the dut
;-----------------------------------------------------------------------------------------
READ
                                ;
        movlw   b'10100000'     ; set slave address and write mode
        movwf   slave
                                ;
        clrf    addr            ; set address to 00h
                                ;
        call    BSTART          ; generate start bit
        nop
        nop
        movf    slave,w         ; get slave address
        movwf   txbuf           ; into transmit buffer
        call    TX              ; and send it
        movf    addr,w          ; get word address
        movwf   txbuf           ; into transmit buffer
        call    TX              ; and send it
        nop
        nop
        call    BSTART          ; generate start bit
        nop
        nop
        movlw   b'10100001'     ; get slave address and read mode
        movwf   txbuf           ; into transmit buffer
        call    TX              ; and transmit it
        nop
        call    RX              ; get 8 bits of data
        bsf     eeprom,do
        call    BITOUT          ; send high ack bit and then a
        call    BSTOP           ; stop bit to end transmission from dut
        nop                     ;
        nop
        nop
        nop
        nop
        bcf     port_a,vcc      ; turn power to dut off
        movlw   .100
        movwf   loops
        call    WAIT            ; wait awhile
        goto    PWRUP           ; go do the whole thing over again
;
END
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN536

Basic Serial EEPROM Operation

Author: Steve Drehobl
Microchip Technology Inc.

Looking for the optimum non-volatile memory product for your system that requires a small footprint, byte level flexibility, low power, and is highly cost effective? Serial EEPROM technology is one of the non-volatile memory technologies that has emerged as a leading embedded control solution. Unfortunately, most system designers are not aware of the serial EEPROM benefits. Also, the supporting documentation in databooks is most often not adequate due to incomplete or ambiguous information. As a result, the system designer often selects a non-volatile solution that does not meet his requirements, or, the designer must face a more complicated design-in with a serial EEPROM.

This article addresses two issues that exist today for designers considering serial EEPROM products:

First, to provide awareness of the application benefits.

Secondly, to provide a primer on the operating principles and instructions. These items are often buried in databook text or not adequately addressed. Also included are common default conditions to significantly reduce the system designer's learning curve.

CONTENTS

Serial EEPROM Applications
Overview of the Primary Protocol Benefits
3-Wire Bus Operation Primer
2-Wire Bus Operation Primer
Microchip 2-Wire Default Conditions
Timing Diagram Attachments

SERIAL EEPROM APPLICATIONS

Serial EEPROMS are ideal non-volatile cost effective memory solutions in applications that require:

- Small footprint and board space as in cellular phone applications
- BYTE level ERASE, WRITE, and READ of data as in a TV tuner
- Low voltage and current for handheld battery applications as in a keyless entry transmitter
- Multiple non-volatile functions in the same application such as a VCR
- Low availability of microcontroller I/O lines

The common applications for Serial EEPROMS are shown below:

Market	Common Applications
Consumer	TV tuners, VCRs, CD players, cameras, radios, and remote controls
Automotive	Airbags, anti-lock brakes, odometers, radios, and keyless entry
Office Automation	Printers, copiers, PCs, and portable PCs
Telecom	Cellular, cordless and full feature phones, faxes, modems, pagers, and satellite receivers
Industrial	Bar code readers, point-of-sale terminals, smart cards, lock boxes, garage door openers, and test measurement equipment

The typical functions that serial EEPROMs are utilized for are:

- Memory storage of channel selectors or analog controls (volume, tone, etc.) in consumer electronics products
- Power down storage and retrieval of events such as fault detection or error diagnostics in automotive products
- Electronic real time event or maintenance logs such as page counting in office automation products. Also, configuration or DIP switch storage in office automation products
- Last number redial storage and speed dial number storage in telecom products
- User in-circuit reprogrammable look up tables such as bar code readers, point-of-sale terminals, environmental controls and other industrial products

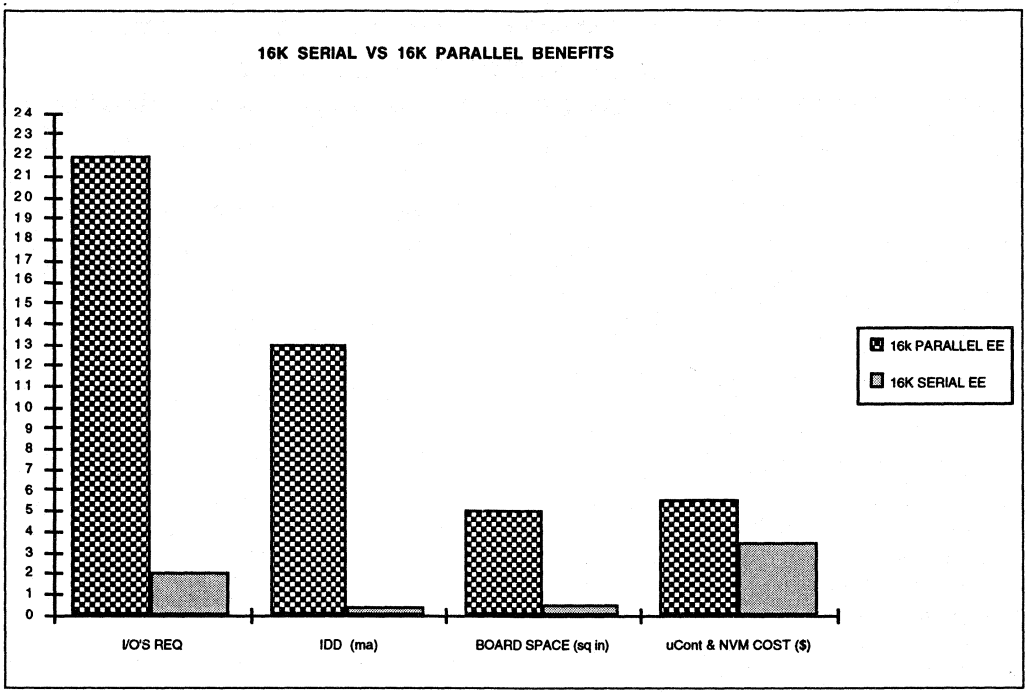
Other application examples include:

- Data storage from a learn function as in a remote control transmitter
- ID number storage for security or remote access for electronic keys and entry databases
- Reprogrammable calibration data for test equipment or analog interface products

7

© 1997 Microchip Technology Inc.

As a result of density and architectural evolution, Serial EEPROMs offer significant benefits in some applications that previously could only utilize Parallel EEPROM products. The diagram below illustrates the footprint and board space differences



The Serial EEPROM requires only 10% of the board space that a Parallel EEPROM requires. Also, the Serial EEPROM requires fewer I/O lines from the microcontroller which significantly reduces the overall system cost and board space.

A very fast READ speed is the only significant limitation of a Serial EEPROM for a decision between a serial and a Parallel EEPROM. It is very interesting to note that the Serial EEPROM READ speed is restricted more by the protocol than the process technology. The 2-wire I^2C (Inter-Integrated Circuit) products must add large internal delays to slow down the part to meet the 100KHz protocol requirements, which will be reviewed later. Characterization of 3-wire bus Serial EEPROMs have indicated clock frequencies in excess of 6MHz.

OVERVIEW OF THE PRIMARY PROTOCOL BENEFITS

After a designer decides to use a serial EEPROM solution, the next step is to select one of the two primary serial EEPROM protocols. Unfortunately, most system designers select the type of serial EEPROM (2- or 3-wire) that they are most familiar with, regardless of the benefits associated with each type.

© 1997 Microchip Technology Inc.

The benefits of each protocol are shown below:

3-Wire Bus Serial EEPROMS	2-Wire Bus Serial EEPROMS
Single VDD supply of <2V to 5.5V	Single VDD supply of <2V to 5.5V
Very low current consumption	Very low current consumption
Reduced overall component cost	Reduced overall component cost
Four pins (other than VCC & GND) are required for operation	Two pins (other than VCC & GND) are required for operation
x16 bit and x8 bit data widths	x8 data bit width
Software WRITE Protection	Hardware WRITE Protection
Edge triggered clocks and signals	Level triggered clocks and signals and input glitch filters for high noise immunity
2MHz+ operation	I^2C standard 100KHz and 400KHz protocols with a 1MHz option
Ready/Busy data polling	Page WRITE capability to 16 bytes
Security options available	Software and hardware compatible from 2K to 16K densities
Less complex protocol	

A 2- wire product is utilized in applications that require an I^2C bus, noise immunity, limited microcontroller I/O pin availability, or a WRITE buffer for multiple bytes to be stored with one instruction. A 3-wire product is utilized in applications that have limited protocol requirements, an SPI protocol, higher clock frequency requirements, or x16 data width applications.

The next two sections describe the basic operation and Microchip's default conditions for the 3-wire and 2-wire Serial EEPROMs to allow the system designer to utilize the benefits of Serial EEPROMs.

3-WIRE BUS OPERATION PRIMER

Many serial EEPROM data sheets are written in a conventional memory data sheet format which emphasizes the features of the part more than the basic operating principles. The operating principles are unfortunately either vaguely embedded in the data sheet text or not included. Serial EEPROMs are not conventional memories due to the Serial communication protocols involved. This section is a PRIMER for the data sheet to familiarize the system designer with the basic principles of the 3-wire bus operation.

Basic Principles

Common device nomenclature is 93XXXX.

The 93XX06 is a 256 bit product.

The 93XX46 is a 1K bit product.

The 93XX56 is a 2K bit product.

The 93XX66 is a 4K bit product.

Four pins are required:

CS (Chip Select)	DI (data in)
CLK (Clock)	DO (data out)

All 93XXXX parts are hardware compatible for these four pins. However, there may be compatibility issues for the other pins.

Even though there is hardware compatibility on the four pins, there can be differences from a software standpoint. Subtle differences between each manufacturer's products, referred to as default conditions, can prevent plug compatibility. These issues are addressed later in the attached 3-Wire Timing Diagram. There is no industry standardized upgrade path for density migration. Please review density upgrades for Microchip's products on a case-by-case basis.

Data is available in x8 or x16 organizations. This selection is determined either by the ORG pin or by purchasing a standard x16 organization.

Units will power-up in a EWDS (ERASE/WRITE Disable State). All ERASE and WRITE functions are disabled until the EWEN (ERASE/WRITE Enable) instruction is performed. This is to prevent accidental data corruption.

An Auto-ERASE (logical "1") cycle is performed during each WRITE Cycle.

The seven instructions are shown in the attached instruction set table. These instructions are for Microchip's 93LCXX family products.

After an instruction is loaded, the CLK and DI pins are in a DON'T CARE state until the next START bit.

© 1997 Microchip Technology Inc.

The following is required for each instruction set (all input bits are triggered by the positive clock edges):

Start Bit	The first Data-in high signal clocked in after CS is high.
Opcode	Two Bits to identify the instruction
Address	Refer to the Instruction Set table for the number of bits required.
Data	Separate data-in and data-out pins. However, these two pins may be tied together for true 3-wire operation. Please refer to the attached 3-wire Bus READ timing diagram example.

READ, WRITE, and ERASE

The attached 93LC66 timing diagrams illustrate the key concepts and timing parameters for each of these operations. Please refer to the instruction set tables and the AC parameters in the databook for supplemental information.

ERASE ALL (ERAL)

An ERASE ALL (ERAL) operation is identified by a "00" opcode. The ERAL instruction requires the next two bits to be clocked in as "10" in the address block of the instruction set. All bits in the array will be set to a logic "1" state by one command in typically less than 10ms.

WRITE ALL (WRAL)

A WRITE ALL (WRAL) operation is also identified by a "00" opcode. The WRAL requires the next two bits to be clocked in as "01" in the address block of the instruction set. The data-in block will contain the data for a SINGLE BYTE which is to be repeated throughout the entire array. For example, if a 4F5A is loaded in the 16 data-in bits of the instruction set, a 4F5A will be written into every word in the array.

EWEN and EWDS

As stated before, all units will power up in to an ERASE/WRITE DISABLE (EWDS) state to prevent data corruption. All future ERASE/WRITE operations must execute an ERASE/WRITE ENABLE (EWEN) opcode until the next power down is detected or until other EWDS opcodes are executed. Please refer to the instruction set table.

© 1997 Microchip Technology Inc.

INSTRUCTION SET FOR 93LC46: ORG = 1 (X16 ORGANIZATION)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	A5 A4 A3 A2 A1 A0	—	D15 - D0	25
EWEN	1	00	1 1 X X X X	—	High-Z	9
ERASE	1	11	A5 A4 A3 A2 A1 A0	—	(RDY/BSY)	9
ERAL	1	00	1 0 X X X X	—	(RDY/BSY)	9
WRITE	1	01	A5 A4 A3 A2 A1 A0	D15 - D0	(RDY/BSY)	25
WRAL	1	00	0 1 X X X X	D15 - D0	(RDY/BSY)	25
EWDS	1	00	0 0 X X X X	—	High-Z	9

NSTRUCTION SET FOR 93LC46: ORG = 0 (X 8 ORGANIZATION)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	A6 A5 A4 A3 A2 A1 A0	—	D7 - D0	18
EWEN	1	00	1 1 X X X X X	—	High-Z	10
ERASE	1	11	A6 A5 A4 A3 A2 A1 A0	—	(RDY/BSY)	10
ERAL	1	00	1 0 X X X X X	—	(RDY/BSY)	10
WRITE	1	01	A6 A5 A4 A3 A2 A1 A0	D7 - D0	(RDY/BSY)	18
WRAL	1	00	0 1 X X X X X	D7 - D0	(RDY/BSY)	18
EWDS	1	00	0 0 X X X X X	—	High-Z	10

INSTRUCTION SET FOR 93LC56: ORG = 1 (X 16 ORGANIZATION)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	X A6 A5 A4 A3 A2 A1 A0	—	D15 - D0	27
EWEN	1	00	1 1 X X X X X X	—	High-Z	11
ERASE	1	11	X A6 A5 A4 A3 A2 A1 A0	—	(RDY/BSY)	11
ERAL	1	00	1 0 X X X X X X	—	(RDY/BSY)	11
WRITE	1	01	X A6 A5 A4 A3 A2 A1 A0	D15 - D0	(RDY/BSY)	27
WRAL	1	00	0 1 X X X X X X	D15 - D0	(RDY/BSY)	27
EWDS	1	00	0 0 X X X X X X	—	High-Z	11

INSTRUCTION SET FOR 93LC56: ORG = 0 (X 8 ORGANIZATION)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	X A7 A6 A5 A4 A3 A2 A1 A0	—	D7 - D0	20
EWEN	1	00	1 1 X X X X X X X	—	High-Z	12
ERASE	1	11	X A7 A6 A5 A4 A3 A2 A1 A0	—	(RDY/BSY)	12
ERAL	1	00	1 0 X X X X X X X	—	(RDY/BSY)	12
WRITE	1	01	X A7 A6 A5 A4 A3 A2 A1 A0	D7 - D0	(RDY/BSY)	20
WRAL	1	00	0 1 X X X X X X X	D7 - D0	(RDY/BSY)	20
EWDS	1	00	0 0 X X X X X X X	—	High-Z	12

INSTRUCTION SET FOR 93LC66: ORG = 1 (X 16 ORGANIZATION

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	A7 - A0	—	D15 - D0	27
EWEN	1	00	11XXXXXX	—	High-Z	11
ERASE	1	11	A7 - A0	—	(RDY/BSY)	11
ERAL	1	00	10XXXXXX	—	(RDY/BSY)	11
WRITE	1	01	A7 - A0	D15 - D0	(RDY/BSY)	27
WRAL	1	00	01XXXXXX	D15 - D0	(RDY/BSY)	27
EWDS	1	00	00XXXXXX	—	High-Z	11

INSTRUCTION SET FOR 93LC66: ORG = 0 (X 8 ORGANIZATION)

Instruction	SB	Opcode	Address	Data In	Data Out	Req. CLK Cycles
READ	1	10	A8 - A0	—	D7 - D0	20
EWEN	1	00	11XXXXXXX	—	High-Z	12
ERASE	1	11	A8 - A0	—	(RDY/BSY)	12
ERAL	1	00	10XXXXXXX	—	(RDY/BSY)	12
WRITE	1	01	A8 - A0	D7 - D0	(RDY/BSY)	20
WRAL	1	00	01XXXXXXX	D7 - D0	(RDY/BSY)	20
EWDS	1	00	00XXXXXXX	—	High-Z	12

© 1997 Microchip Technology Inc.

2-WIRE BUS OPERATION PRIMER

As indicated in the 3-wire bus section, many serial EEPROM data sheets are written in a conventional memory data sheet format which emphasizes the features of the part more than the basic operating principles. The operating principles are, unfortunately, either vaguely embedded in the data sheet text or not included. This section is a PRIMER for the data sheet to familiarize the system designer with the basic 2-wire serial EEPROM operation principles.

Basic Principles

The common device nomenclature is 24XXXX and 85XXXX.

Only the SCL and SDA pins are essential for bus operation. The other pins are supplementary:

SCL (Serial clock)

SDA (Serial Data)

WP (Active High WRITE Protection)

A0, V A1, and A2 (Chip or block select)

SDA's open-drain requires a pull-up resistor to V_{DD}.

The data is organized as x8.

Signals are level triggered, not edge triggered. Also, there are filters on the inputs that will filter noise glitches <100ns wide.

An Auto-ERASE (logical "1") cycle is performed during each WRITE cycle.

The I^2C protocol utilizes master/slave bi-directional communication. A device that sends data onto the bus is defined as the transmitter, and a device that is receiving data is the receiver. Both the master and the slave can operate as the transmitter or receiver. The bus must be controlled by a master device (most often a microcontroller), which generates the serial clock (SCL), controls the bus direction, and generates the START and the STOP conditions.

The serial EEPROM is the slave. The serial EEPROM will be the bus transmitter during READ operations and when the serial EEPROM must acknowledge data transmitted by the master.

START and STOP bits control the bus activity. Operations must begin with a START bit and end with a STOP bit.

A START bit is when SDA transitions LOW while SCL is HIGH while observing the START set-up and hold time specifications.

A STOP bit is when SDA transitions HIGH while SCL is HIGH while observing the STOP set-up and hold time specifications.

Data is recognized as valid while SCL is high. The data on SDA must observe data in set-up and hold specifications before and after SCL is pulsed. There is only one bit of data for each SCL pulse.

Control Byte Requirements

After a START bit, each command begins with an 8 bit control byte sent by the master. This control byte has the following three primary functions before the data and/or word address information is loaded for all commands:

Identify the serial EEPROM as the slave addressed on the bus.

Select the specific serial EEPROM or the internal memory block on the bus. There may be up to 8 serial EEPROMs on the bus)

Select the READ or WRITE function for the next command transmitted by the master.

The diagram of a control byte (not including the START bit) is shown below:

1	0	1	0	A2	A1	A0	X
I^2C Slave Address				Chip or Block Select			Read or W rite bit

Control Bits 1-4 are the Slave Address Bits (Must be 1010 for Memory)

Since there is not a chip select pin, the part is selected by a four bit code in the control byte to identify the type of product. The four bit code was established by Philips for the I^2C protocol. A 1010 code identifies the slave device as a Serial EEPROM. The Serial EEPROM will remain in stand-by until the 1010 code is transmitted on the bus. Other non Serial EEPROM slave devices will not respond to the 1010 code on the bus.

Control Bits 5-7 are the 1 of 8 Chip or Block Address Select Bits

The next three control bits are utilized for the chip selection or internal block selection. The standard I^2C protocol was developed to allow up to 16K bits of memory to be selected. This could be accomplished by accessing a combination of devices or blocks within a device, as shown in the table on the following page:

© 1997 Microchip Technology Inc.

Device	K bits Density	Internal Blocks	A0	A1	A2	Bus Access Devices
24LC01B, 24C01,85C72	1	1	H or L	H or L	H or L	Up to 8 devices
24LC02B, 24C02,85C82	2	1	H or L	H or L	H or L	Up to 8 devices
24LC04B, 24C04,85C92	4	2	X	H or L	H or L	Up to 4 devices
24LC08B	8	4	X	X	H or L	Up to 2 devices
24LC16B	16	8	X	X	X	Only 1 device

X= NOT USED. This pin must be tied to VSS or VDD. It is not recommended to FLOAT these pins since there may be test modes accessed to these pins via a high voltage signal.

These three bits for this select must match the hardware conditions (IF ANY ARE USED) of the external A0, A1, and A2 pins or the internal block selects.

With this selection scheme, devices from 2K to 16K are software compatible. For example, four 2K devices or one 8K device could be connected to the bus with the same software.

The A0, A1, and A2 signals are the same for the 1K and 2K products. The A7 bit for the 1K product is a DON'T CARE.

The A0, A1, and A2 pins are not commonly used today in the industry with the advent of the density evolution up to the I^2C protocol limit of 16K bits.

Control Bit 8 Operation Code

If this bit is a "1" then the operation will be a READ

If this bit is a "0" then the operation will be a WRITE

After the control byte acknowledge bit is generated by the serial EEPROM, the master will send the appropriate word address and data information.

Acknowledge Requirements

The serial EEPROM must generate an acknowledge bit after receiving each byte segment in a command. The serial EEPROM will generate the acknowledge bit automatically after the master has transmitted all of the data for the segment.

To acknowledge the master, the serial EEPROM must pull the SDA line LOW during the entire HIGH period of the next clock generated by the master. During the READ operations, the master must acknowledge each data byte or the serial EEPROM will abort the READ operation and return to a stand-by mode waiting for the next START bit.

The attached 24LC16 timing diagrams illustrate the READ and WRITE operations.

MICROCHIP 2-WIRE DEFAULT CONDITIONS

As stated before, data sheets do not provide adequate information on basic operation. This lack of information forces each reader of the databook to make interpretations about the operating conditions. These readers have included other semiconductor circuit designers, which unfortunately leads to subtle compatibility problems. The part is designed to operate to the default of the circuit designer's interpretation. This next section details Microchip's default conditions to help the system engineer minimize "Trial and Error" prototyping and to increase the awareness of these default conditions.

Also, to improve corporate-wide compatibility, Microchip is standardizing their circuits on various product versions. Unless indicated otherwise, all references to default conditions are for the 24LCXX products, not the 24CXXXX products.

Power Up

READ, WRITE, and ERASE operations are valid 5 µS after VDD has ramped to the specified operating range.

PAGE WRITE by Product for Multiple BYTE WRITE Operation

The 24C01 and 24C02 have a 2-byte buffer.

The 24C04 has an 8-byte buffer.

The 24LC01 and 24LC02 have an 8 byte page.

The 24LC04, 24LC08, and 24LC16 have a 16 byte page.

The buffer will load bytes identically as the page loads bytes. The difference in the two modes is that the buffer will execute a WRITE of one byte per WRITE cycle in sequence. The page mode will execute all bytes loaded in one WRITE cycle in parallel.

© 1997 Microchip Technology Inc.

There are pages within blocks. For a 16 byte page product, the most significant 4 bits of the word address point to the page address and the least significant 4 bits point to the byte address within a page. For an 8 byte page product, the most significant 5 bits of the word address point to the page address and the least significant 3 bits point to the byte address within a page.

The number of bytes loaded in to the page is from one byte up to the page size. For example, three bytes can be loaded into the 16 byte page of the 24LC16. If during the loading of the fourth byte a STOP bit is received, the page will WRITE three bytes. The fourth byte will not be written since loading the fourth byte was not complete.

Note: New versions released in March 1993 will default to ABORTING the entire operation if a STOP bit is received in the middle of a byte while loading a page.

If more than 16 bytes are loaded in the page of a 16 byte page product, then the 17th byte will override the data loaded into the original first byte (the page data will wrap around WITHIN a page). Therefore, the system designer must take precautions to not WRITE over a page boundary during a multiple byte WRITE operation.

Bytes not changed in the page will NOT result in data corruption in the array. For example, If two bytes are loaded in to the 24LC16 page with the least significant word address bits of 0000 and then a STOP bit is transmitted. Bytes 1 and 2 in the array will have the data changed to the new page contents. Bytes 3 through 16 WILL NOT change.

The WRITE operation will not be executed until a STOP bit is transmitted.

At this point, the serial EEPROM is free from the bus since the actual WRITE function is self-timed. Therefore, the microcontroller interfacing to the serial EEPROM may perform other functions not associated to communication with the serial EEPROM during the self-timed WRITE operations.

Once the part is in the auto-ERASE mode, it will complete the ERASE/WRITE operation unless power is removed. STOP and START bits will be ignored.

READ

Once the Serial EEPROM is in a RANDOM READ operation, it can be placed into the sequential READ operation. If the master issues an acknowledge bit instead of a STOP bit, the Serial EEPROM will READ the next sequential 8 bits. The Serial will wait for the next bit command from the master. The sequential READ will continue as long as the master issues an acknowledge bit on the next clock cycle after the last bit is READ. The READ will continue from block to block and will wrap around if the last bit in the array is addressed. Again, this will continue until the master issues a STOP bit instead of an acknowledge bit.

While reading zeroes the master cannot pull SDA high to generate a STOP bit, since the serial EEPROM SDA pin is outputting a low. To recover from a fault during a READ, repeat 9 clocks with data floating high. Therefore, the acknowledge bit will not occur and the part will reset and return to stand-by.

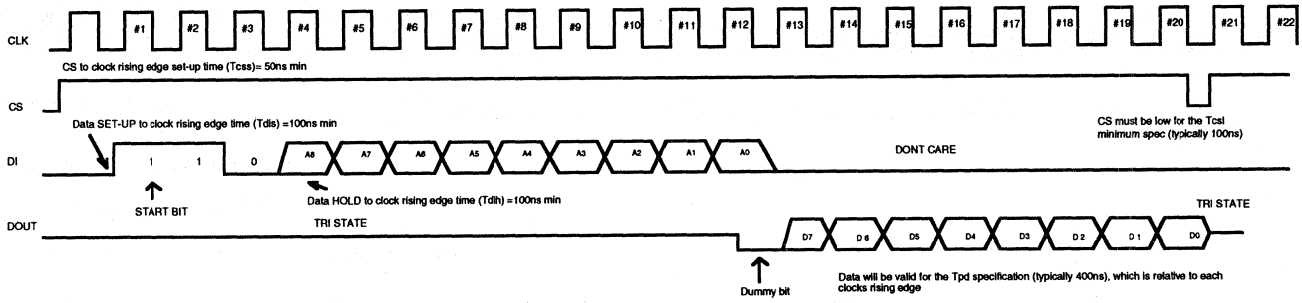
A START bit during an operation will cease the current operation and begin the next operation.

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

3 WIRE BUS EXAMPLE

MICROCHIP 93LC66 READ CYCLE TIMING DIAGRAM



NOTE: THIS EXAMPLE IS FOR X8 OPERATION OF MICROCHIP'S 93LC66. INSTRUCTION LENGHTS MAY VARY WITH ARRAY SIZE AND DATA WIDTHS

Data must conform to specified set-up and hold times (Tdis and Tdih) relative to the RISING clock edge. Each parameter is typically 100ns.

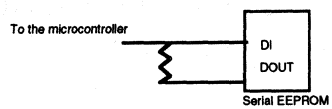
A Read operation is identified by the "1 0 " op-code following the start bit.

Next, the address location bits are loaded.

Then the address contents are clocked out on the rising clock pulse edge. Data will become valid on the DOUT pin per the specified Tpd time (typically 400ns) relative to the rising edge of the clock on the DOUT pin. Note, the first bit output will be a "dummy bit" with a logical zero state. This event is triggered by the clock rising edge of the last address bit for a duration of one clock pulse.

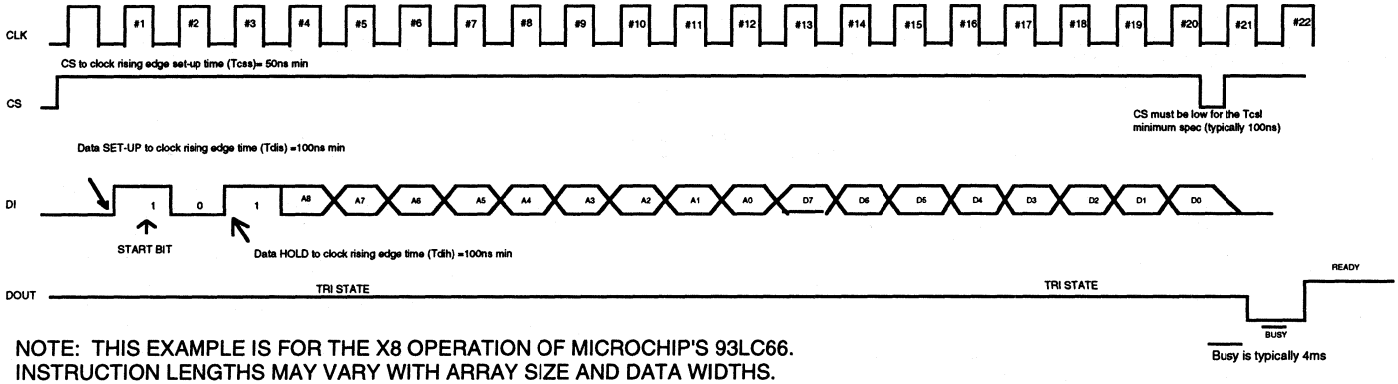
If the data from the current address is complete and the clock pulses continue, the data from the next address will be READ automatically as long as CS remains high. This is the SEQUENTIAL READ FUNCTION. READ operations will continue while clock pulses continue or until CS is brought low.

It is possible to tie the DOUT pin and the DI pin together to save on I/O requirements from the microcontroller. Caution must be exercised to avoid bus contention for an A0 high condition, because of the dummy bit. It recommended that a resistor between the microcontroller port connected to the DI pin and DOUT pin be added for isolation. This example is shown below:



3 WIRE BUS EXAMPLE

MICROCHIP'S 93LC66 WRITE CYCLE TIMING DIAGRAM



NOTE: THIS EXAMPLE IS FOR THE X8 OPERATION OF MICROCHIP'S 93LC66.
INSTRUCTION LENGTHS MAY VARY WITH ARRAY SIZE AND DATA WIDTHS.

Data must conform to specific set-up and hold times (Tdis and Tdih) relative to the clock edge. Each parameter is typically 100ns.

A WRITE operation is identified by a the "0 1 " op code following the start bit

Next, the address location bits are loaded on the DI pin.

Then, the data bits to be written are loaded on the DI pin.

CS must be brought low after the last DI bit is loaded. When CS is brought low for the Tcsl period, a self timed WRITE is executed. If too many bits are loaded during ERASE and WRITE instructions prior to CS being brought low at the end of an instruction set, then the extra bits will be ignored. Only the first bits loaded will be executed.

The DOUT pins only function during a WRITE is to indicate the status of the write with the READY/$\overline{\text{BUSY}}$ function. While DOUT is low, the Serial EEPROM is indicating that programming is not complete (the part is BUSY). When DOUT is high,the Serial EEPROM is indicating that programming is complete and it is READY for another instruction. Note CS must be brought high after completing the Tcsl time is complete to initiate the READY/ BUSY function. Microchip's 93LCXX products can be polled multiple times for the same cycle.

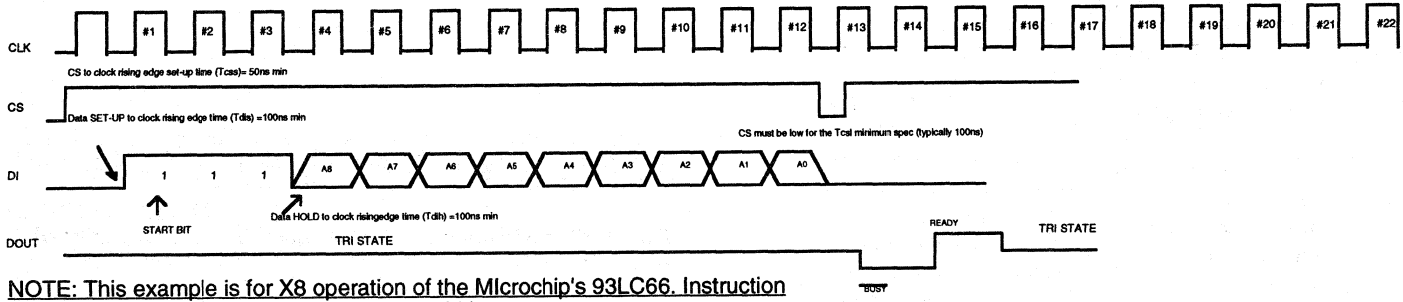
Up through clock pulse 20, data for the instruction is being LOADED. When the CS goes low, the instruction is being EXECUTED. If there are not enough bits loaded during ERASE and WRITE instructions prior to CS being brought low, then the operation WILL NOT BE EXECUTED and the Serial EEPROM will return to stand-by.

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

3 WIRE BUS EXAMPLE

MICROCHIP 93LC66 ERASE CYCLE TIMING DIAGRAM



NOTE: This example is for X8 operation of the MIcrochip's 93LC66. Instruction lengths may vary with array size and data widths.

Data must conform to specified set-up and hold times (Tdis and Tdih) relative to the clock edge. Each parameter is typically 100ns.

AN ERASE operation is identified by a "11 " two bit code that follows the start bit

Next, the address location bits are loaded on the DI pin.

THERE ARE NO DATA BITS TO LOAD. THE ADDRESS LOCATION LOADED WILL BE SET TO AN ERASE STATE OF "1".

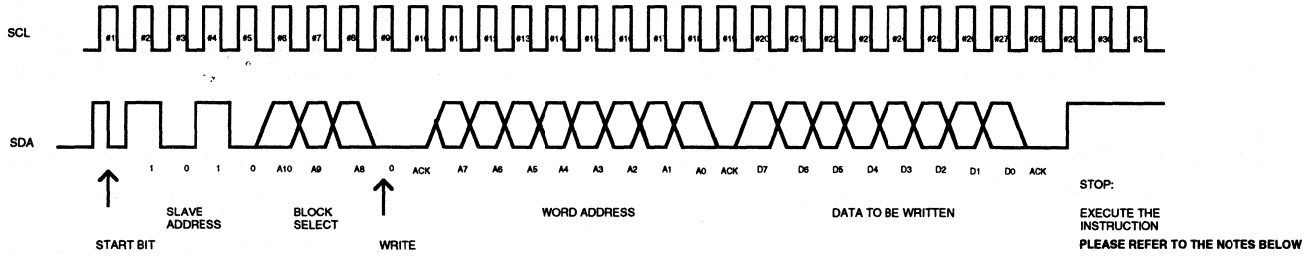
CS must be brought low after the last DI bit is loaded. When CS is brought low for the Tcsl period, a self timed ERASE is executed. If too many bits are loaded during the ERASE and WRITE instructions prior to CS being brought low at the end of an instruction set, then the extra bits will be ignored. Only the first bits loaded will be executed.

The DOUT pin's only function during a ERASE is to indicate the status of the write with the READY/$\overline{\text{BUSY}}$ function. While DOUT islow, the Serial EEPROM is indicating that programming is not complete (the part is BUSY). When Do is high, the Serial EEPROM is indicating that programming is complete and it is READY for another instruction. Note CS must be brought high after completing the Tcsl time is complete to initiate the READY/$\overline{\text{BUSY}}$ function.

Up through clock pulse 12, the address for the instruction is being LOADED. When CS goes low, the instruction is being EXECUTED.If there are not enough bits loaded during the ERASE and WRITE instructions prior to CS being brought low, then the operation WILL NOT BE EXECUTED and the serial EEPROM will return to stand-by.

2 WIRE BUS EXAMPLE

MICROCHIP 24LC16 BYTE WRITE CYCLE TIMING DIAGRAM



NOTE THE SDA POSITION OF THE START AND STOP BITS. THE SDA TRANSITION IS DURING A HIGH SCL PULSE

ALL OTHER BITS TRANSMITTED MUST COMPLY WITH THE 100KHZ CLOCK IIC PROTOCOL DATA SET -UP TIME OF 250NS (TSU: DAT) FOR DATA TO BE ESTABLISHED PRIOR TO THE RISING CLOCK EDGE AND THE HOLD TIME OF 0NS (THD:DAT) FOR THE FALLING CLOCK EDGE.

THE STOP BIT ON CLOCK PULSE #29 WILL INITIATE A SELF TIMED WRITE. **THE SERIAL EEPROM CAN NOT EXECUTE ADDITIONAL INSTRUCTIONS UNTIL THE CYCLE IS COMPLETE.**

TO EXECUTE A PAGE WRITE, CONTINUE TO LOAD 8 BITS OF DATA AT CYCLE 29 INSTEAD OF ISSUING A STOP BIT (FROM THE MASTER). REMEMBER , A CLOCK PULSE MUST BE ALLOCATED AFTER EACH SUBSEQUENT 8 BITS FOR THE SERIAL EEPROM TO ISSUE AN ACKNOWLEDGE SIGNAL (LOW). AFTER THE DESIRED NUMBER OF BYTES HAVE BEEN LOADED, UPTO THEIR PAGE SIZE, THE MASTER MUST ISSUE A STOP BIT TO EXECUTE THE INSTRUCTION.

THE SEQUENCE OF EACH WRITE COMMAND AND THE MASTER/SERIAL DIRECTION OF THE COMMUNICATION IS SHOWN BELOW :

BYTE WRITE

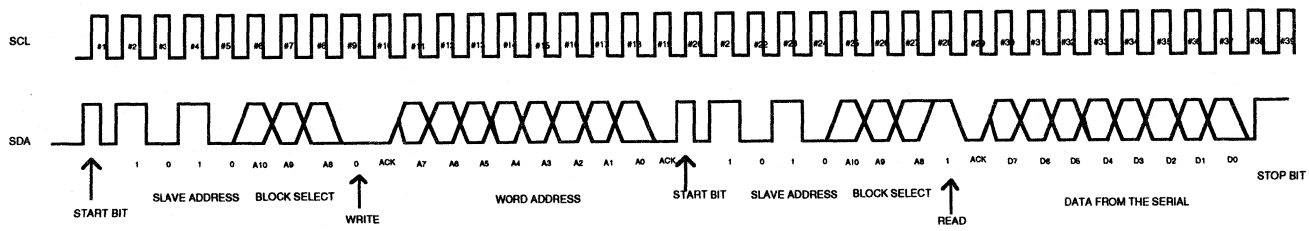
START BIT FROM THE MASTER	
CONTROL BYTE FROM THE MASTER	
	ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER	
	ACKNOWLEDGE BIT FROM THE SERIAL
DATA BYTE FROM THE MASTER	
	ACKNOWLEDGE BIT FROM THE SERIAL
STOP BIT FROM THE MASTER	

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

2 WIRE BUS EXAMPLE

MICROCHIP 24LC16 READ CYCLE TIMING DIAGRAM



NOTE : THE FIRST 19 CLOCK PULSES OF THE WRITE COMMAND ARE IDENTICAL TO THE FIRST 19 CLOCK PULSES IN THE READ COMMAND. EVEN THE 9TH CLOCK PULSE IS A WRITE BIT "0" TO TRANSMIT TO THE SERIAL EEPROM THE DESIRED WORD ADDRESS.

THE READ COMMAND HAS TWO START BITS. THIS IS FOR THE RANDOM READ COMMAND.

AS SHOWN ON THE PREVIOUS PAGES, IF A READ IS DESIRED FROM A CURRENT ADDRESS THEN THE FIRST 19 CLOCK PULSES ARE NOT REQUIRED. THEREFORE, THE FIRST START BIT IS AT CLOCK PULSE #20. THIS IS ONLY FOR THE CURRENT ADDRESS READ COMMAND

ANOTHER USEFUL READ COMMAND IS THE SEQUENTIAL READ COMMAND. THE SEQUENTIAL READ COMMAND IS THE SAME AS THE RANDOM READ COMMAND; HOWEVER, THE MASTER MUST ISSUE AN ACKNOWLEDGE BIT INSTEAD OF THE STOP BIT AS SHOWN FOR CLOCK PULSE #38. THIS SIGNALS THE SERIAL TO READ THE DATA FROM THE NEXT SEQUENTIAL ADDRESS. THE MASTER MUST CONTINUE TO ACKNOWLEDGE EACH BYTE RECEIVED UNTIL THE MASTER ISSUES A STOP BIT.

NOTE THE SDA POSITION OF THE START AND STOP BITS. THE SDA TRANSITION IS DURING A HIGH SCL PULSE

ALL OTHER BITS TRANSMITTED MUST COMPLY WITH THE 100KHZ CLOCK IIC PROTOCOL DATA SET -UP TIME OF 250NS (TSU: DAT) FOR DATA TO BE ESTABLISHED PRIOR TO THE RISING CLOCK EDGE AND THE HOLD TIME OF 0NS (THD:DAT) FOR THE FALLING CLOCK EDGE.

READ (FROM A RANDOM ADDRESS)

START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R / W = 0)
ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R / W = 1)
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
STOP BIT FROM THE MASTER

READ (FROM THE CURRENT ADDRESS)

START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R / W = 1)
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
STOP BIT FROM THE MASTER

READ (Sequential READ of 3 bytes)

START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R / W = 0)
ACKNOWLEDGE BIT FROM THE SERIAL
WORD ADDRESS FROM THE MASTER
ACKNOWLEDGE BIT FROM THE SERIAL
START BIT FROM THE MASTER
CONTROL BYTE FROM THE MASTER (R / W = 1)
ACKNOWLEDGE BIT FROM THE SERIAL
DATA FROM THE SERIAL
ACKNOWLEDGE BIT FROM THE MASTER
DATA FROM THE SERIAL
ACKNOWLEDGE BIT FROM THE MASTER
DATA FROM THE SERIAL
STOP BIT FROM THE MASTER

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN537

Serial EEPROM Endurance

Author: *Steve Drehobl*
Microchip Technology Inc.

INTRODUCTION

The term "endurance" has become a confusing parameter for both users and manufacturers of EEPROM products. This is largely because many semiconductor vendors treat this important application-dependent reliability parameter as a vague specmanship topic. As a result, the system engineer often designs without proper reliability information or under-utilizes the EEPROM as an effective solution.

Endurance (the number of times an EEPROM cell can be erased and rewritten without corrupting data) is a measure of the device's reliability, not its parametric performance. As such, endurance is not achieved by somehow making EEPROM devices more durable or robust to extend the life of the intrinsic erase/write cycle, but rather by reducing their defect-density failure rates. This has a direct impact on the design engineer characterizing EEPROM memory needs for an application and evaluating components from various manufacturers. The system design engineer needs to understand not only the relationship between the application, expected use and failure mechanisms, but also how the manufacturer has arrived at published endurance data for its components.

This tutorial volume is intended to clarify some of the issues in the industry and provide a tool for the system design engineer, the system reliability engineer, and the component engineer to determine EEPROM reliability and understanding how to apply it to actual application requirements. It will examine four main areas:

- CMOS floating gate memory cell operation and characteristics
- Significant process and design interactions and endurance characterization variables
- Common misinterpretations of endurance
- Determining some real world application reliability requirements

EEPROM MEMORY CELL OPERATION AND CHARACTERISTICS

In discussing endurance characteristics of EEPROMs, it's important to review how these components operate, and why and how they fail. Figure 1 illustrates a CMOS floating gate EEPROM cell, including voltage conditions for READ, ERASE, and WRITE operations. To erase or write, the row select transistor must have the relatively high potential of 20V. This voltage is internally generated on chip by a charge pump, with the only external voltage required being VDD. The only difference between an ERASE and a WRITE is the direction of the applied field potential relative to the polysilicon floating gate.

When 20V is applied to the polysilicon memory cell gate and 0V is applied to the bit line drain (column), electrons tunnel from the substrate through the 90-angstrom Tunnel Dielectric (TD) oxide to the polysilicon floating gate until the polysilicon floating gate is saturated with charge. The cell is now at an ERASE state of "1". When 0V is applied to the polysilicon memory cell gate and 20V is applied to the bit line drain (column), electrons tunnel from the polysilicon floating gate through the TD oxide to the substrate. The cell then is at a WRITE state of "0". This sequence of the transfer of charge onto the floating gate (ERASE) and the electrical removal of that charge from the floating gate (WRITE) is one ERASE/ WRITE cycle, or "E/W cycle."

The field (applied voltage to an oxide thickness) across the tunneling path created by the 20V potential is extremely high in order to transfer the electrons. Over the cell's "application time," as measured by E/W cycles, the EEPROM cell begins to wear out due to the field stress. The EEPROM cell wears out as the number of cycles increase resulting in the voltage margin between the ERASE and WRITE states decreasing until finally there is not enough margin for the EEPROM sense amp to detect a difference in the two states during a READ. Failure is defined as when the sense amp can no longer reliably differentiate logic state changes.

Figure 2 (single cell EEPROM endurance characteristics) illustrates that the intrinsic wear out point for a normal cell with specified dimensions and electrical characteristics is very acceptable, in excess of 2 million E/W cycles. Failures at lower cycles are due mostly to very small defects or imperfections in the oxide or silicon-to-oxide interface.

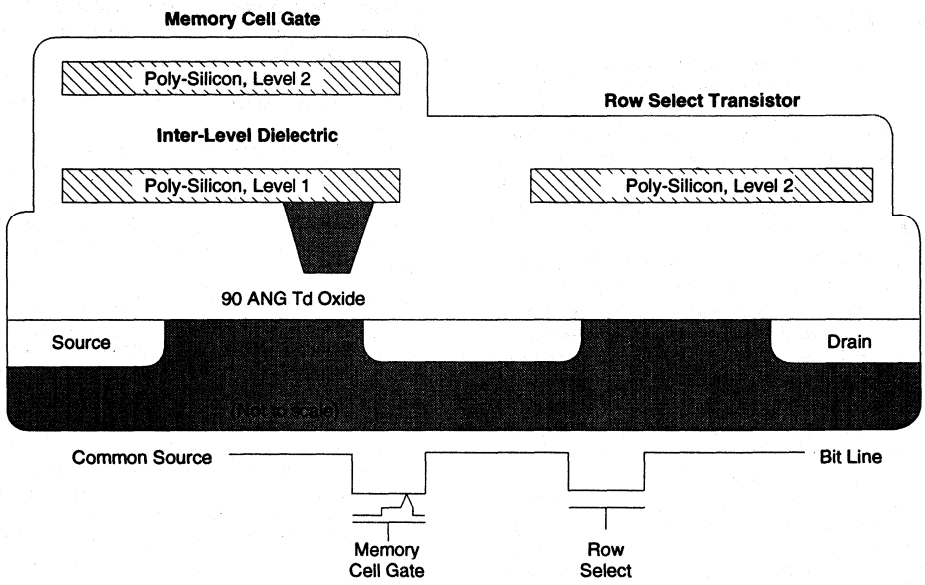
© 1997 Microchip Technology Inc.

A key point to remember is that most failures occurring at less than 2 million E/W cycles are due to the number of defects per a given area (defect density dependent.) Thus high EEPROM endurance reliability is achieved by reducing the defect density failure rates, not by increasing the number of intrinsic cycles in the cell's operational design.

Error correction circuits are design techniques commonly used by EEPROM manufacturers to increase endurance by reducing the failure rate caused by single bit failures. These circuits are transparent to the user. One typical scheme is using 4 bits of error correction for every 8 "real" bits (one byte). In this scheme, one bit failure in the byte is correctable, while if two bits within the byte fail, the byte is not correctable.

Another error correction scheme is to use one "parity" bit for every "cell." Here both EEPROM cells must fail to result in a bit fail.

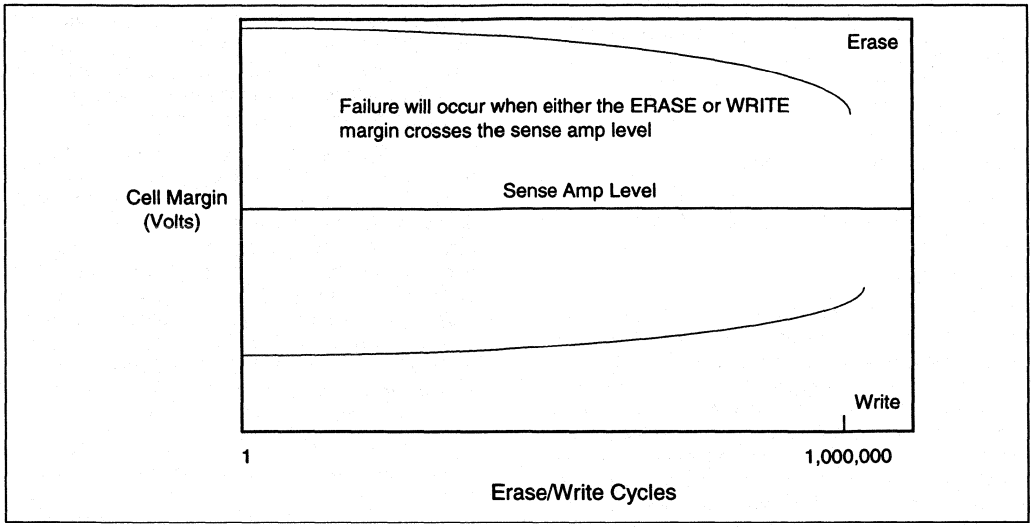
FIGURE 1: CMOS FLOATING GATE EEPROM CELL



	Read	Write	Erase	Standby
Bit Line	1.6 volts	18.0 volts	0.0 volts	0.0 volts
Row Select Gate	5.0 volts	20.0 volts	20.0 volts	0.0 volts
Memory Cell Gate	5.0 volts	0.0 volts	20.0 volts	0.0 volts
Common Source	0.0 volts	Float	0.0 volts	0.0 volts

© 1997 Microchip Technology Inc.

FIGURE 2: SINGLE EEPROM CELL ENDURANCE CHARACTERISTICS



PROCESS AND DESIGN VARIABLES AFFECTING ENDURANCE

There are many subtle process and design variables that have a strong impact on endurance. These interacting variables will play very different roles depending on the different process technologies of various semiconductor manufacturers.

The primary interaction is the amount of TIME at the HIGH VOLTAGES that is ultimately applied to the cell. A finite amount of time at finite voltages are required to achieve "optimal" ERASE and WRITE thresholds. If the time is too short and the voltage is too low, the EEPROM will not program to the proper threshold. Also, if the programming ramp time is too fast and the voltage is too high, the EEPROMs endurance will be reduced. Unfortunately, there is most often a trade-off between fast reliable programming performance or high endurance reliability. Some of the significant process and design variables are shown below and their impact on programming performance and endurance performance.

PARAMETER	PROGRAMMING	ENDURANCE
Internal High Voltages	HIGHER = Faster Programming	LOWER = Increased Endurance
Internal High Voltage Ramp Rate	FASTER = Faster Programming	SLOWER = Increased Endurance
Programming Time	LONGER = Improved Voltage Margin on the Cell	SHORTER = Less Oxide Stress for Increased Reliability
TD Oxide Thickness	THICKER = Slower Programming	THINNER = Reduced Endurance
Temperature	LOWER = Faster Programming	LOWER = Increased Endurance

© 1997 Microchip Technology Inc.

COMMON MISINTERPRETATIONS

In examining industry EEPROM literature on the topic of endurance, it's easy to misunderstand or misinterpret endurance concepts due to incomplete databook statements. The following are clarifications to some of the common misinterpretations:

Endurance and Read Cycles

READ operations are unlimited since they impose virtually no stress on the cell. Endurance data apply only to E/W cycles.

Erase/Write Ratings

E/W ratings are based on each byte in the application, not on the number of opcodes or control byte commands utilized. For example, if a part is rated to 100K E/W cycles, then each individual byte can be erased and written 100K times. The part is NOT limited to only a total of 100K E/W opcodes or control bytes. This is probably the most common misinterpretation made by system designers. Endurance is thus an interactive application-specific reliability parameter. It is not a typical data sheet specification, such as a parametric AC/DC specification with benchmark standards for measurement.

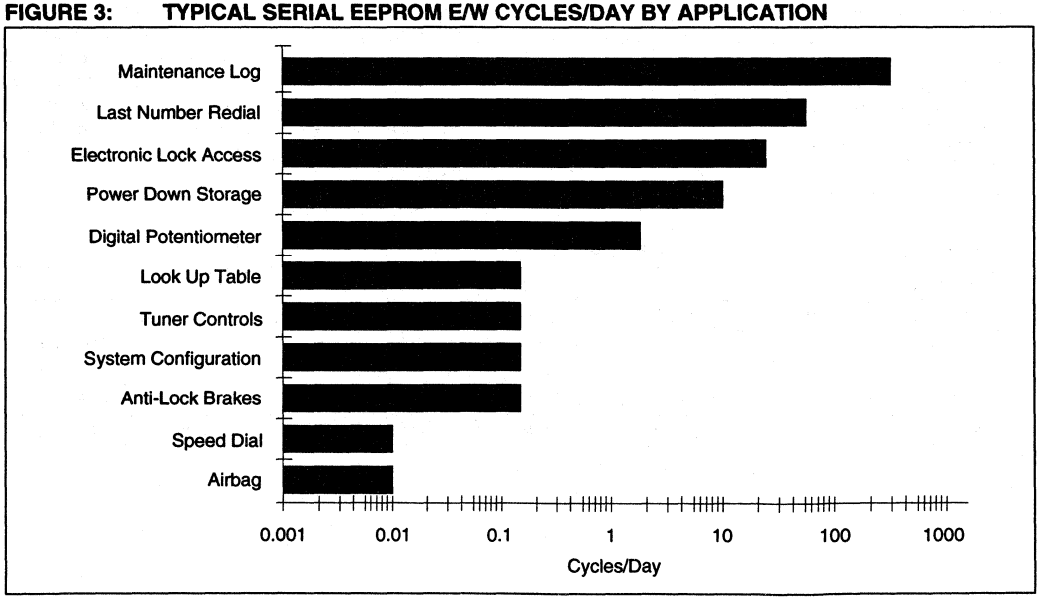
Cycles/Day

In many cases, a serial EEPROM is used for widely varying functions in an application. These functions have different E/W usage requirements (cycles/day), resulting in different endurance requirements and, usually, different reliability results for each function.

For example, assume a given end-product application will have a 10-year life. For each function within that application, an assumption must be made for the expected E/W cycles per day for a given segment of bytes. If a function has a segment of bytes cycled 1 time per day, then this segment of bytes will have 3,650 cycles in its lifetime (365 days per year for 10 years at 1 cycle per day and 7 days per week operation). Any given segment of bytes would have to cycle 274 times per day everyday to reach 1,000,000 E/W cycles in its 10-year application lifetime. Such a frequency is, of course, very rare in actual applications.

For reference purposes, Figure 3 indicates typical cycles per day for some common applications. Although many manufacturers routinely discuss very high numbers of E/W cycles, the amount of applications actually utilizing 1 million cycles is very small.

FIGURE 3: TYPICAL SERIAL EEPROM E/W CYCLES/DAY BY APPLICATION



© 1997 Microchip Technology Inc.

A further and very important incorrect assumption often made is that ALL bits in an application need the same number of cycles and endurance ratings. In most applications, however, functions that require a high number of E/W cycles per day require only a small number of bits. Last number redial in a telephone, for example, consumes many E/W cycles per day, but utilizes only a few bytes for this function. By contrast, speed dial storage in that same telephone consumes only a fraction of E/W cycles per day, but requires a relatively large segment of bits to accommodate the many speed dial options. In such an application, the same serial EEPROM normally performs both functions at different address locations.

ENDURANCE DATA FROM THE CUSTOMER'S PERSPECTIVE

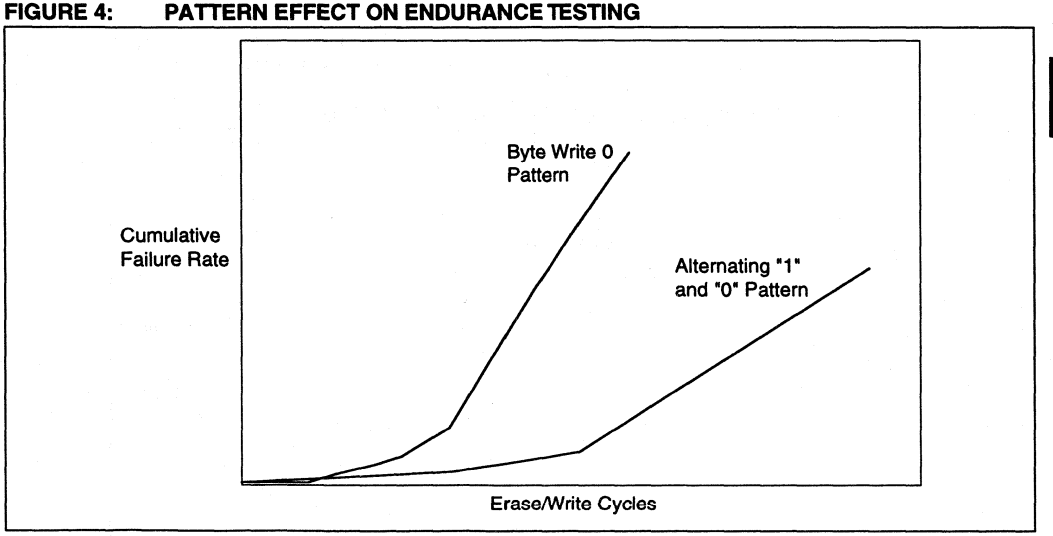
Unfortunately, an industry standard for an endurance test method has yet to be adopted. Since endurance data is not baselined, the process of evaluating endurance becomes that much more complicated for the system designer and reliability engineer.

It is not uncommon for customers to request endurance data from many semiconductor vendors. All vendors would be expected to comment that they experience a low failure rate through 100K E/W cycles. While this can be a true statement, it can also be a very incomplete statement. It is extremely doubtful that all vendors test their components to the same conditions. Yet the variables within endurance testing are extremely significant. Small differences in text protocol can have enormous differences. Pattern, cycling mode, temperature and array size, for example, are the most significant testing variables.

First, memory cell failure rates are defect density driven up to the intrinsic wear out point. Existing defects in a cell, while not causing failure initially, are stressed during every transfer of electrons through the TD oxide until they eventually cause cell failure. Worst case testing would be to erase and write each bit, which is what a write all "0"s pattern with an auto-erase of "1" routine will perform. Indeed, this write all "0"s test pattern will produce very different results than a checkerboard test pattern of alternating "1"s and "0"s within a byte, since cells are changed more often writing all "0"s than in an alternating "1" and "0" write pattern. The resultant failure rate differences are indicated on the pattern effect graph in Figure 4.

In actual use, however, a system will experience a random pattern much more like the alternating "1"s and "0"s pattern than the more stressful all "0"s pattern. The key point for system designers is to determine how accurate a test routine has been used to determine a particular manufacturer's endurance data, and make the appropriate judgement on that part's expected endurance in the application.

FIGURE 4: PATTERN EFFECT ON ENDURANCE TESTING



7

Second, the cycling mode graph in Figure 5 indicates that significantly different results can be achieved in endurance testing using a block cycle mode than using a byte cycle mode. The block mode is commonly used by manufacturers to "speed up" the endurance characterization process. However, endurance results usually will appear much better for the block mode than the byte mode, due to the high voltage variables discussed earlier. The reason is that the voltage ramp rate is significantly SLOWER, the high voltages are slightly lower, thus less stressful for block cycling since the capacitive load of the entire array is on the high voltage charge pump. The capacitive load is much lower with a single row or byte, which thus has a significantly faster ramp rate. Also, there is not a polysilicon to polysilicon stress for adjacent cells since all cells are at the same potential. Most often, these factors combine to yield lower failure rates for block cycling than for byte cycling. Again, the test conditions must match the system conditions.

Finally, increasing temperature also increases stress on the cell. Microchip's endurance characterization data indicates that increasing temperature adds an activation energy (Ea) of 0.12eV on the cell. From a 25°C to 85°C ambient the acceleration factor is approximately 2.1. Therefore, the higher the temperature, the higher the stress. These results will vary significantly with each EEPROM manufacturer.

RECOMMENDED EEPROM ENDURANCE TESTING

Microchip believes that EEPROM components should be endurance tested to reflect system conditions. Therefore, units are cycled to an alternating ONE and ZERO pattern (checkerboard), then to an alternating ZERO and ONE pattern (inverse checkerboard).

Again, since endurance characterization data indicates that a random single bit fail is the primary first order failure mode, endurance is defect density (def/cm2 or segment of bit size) dependent. Therefore an expected failure rate range by density can be established.

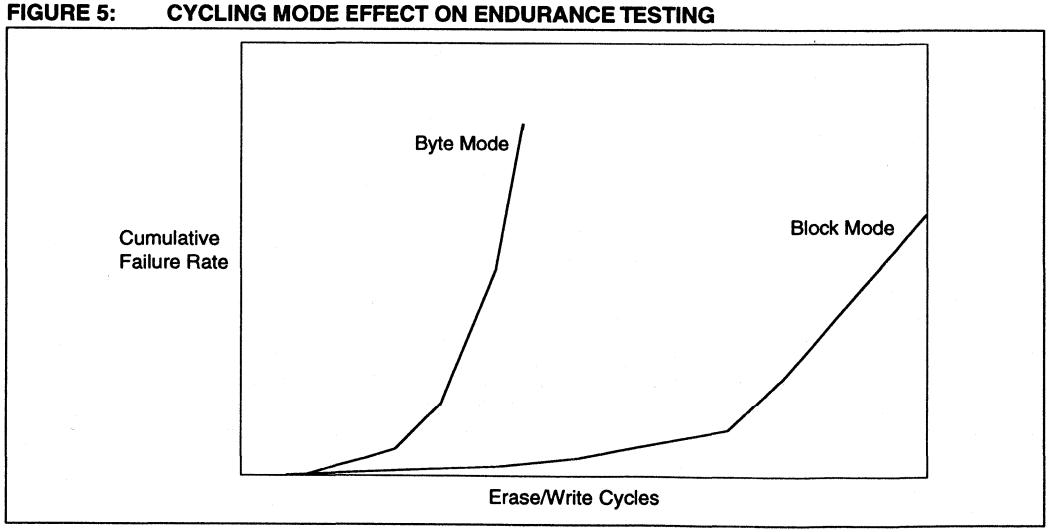
DETERMINING THE RELIABILITY CALCULATIONS

There are three primary components for the system design engineers to use in determining the endurance reliability required for a defect density limited application. These three components are:

- Erase/write cycles/day estimated for the function.
- The number of bits in the function (or segment size).
- Case operating temperature of the Serial EEPROM.

Let's look at three typical examples utilizing the above information to predict a cumulative failure rate at different points in a system lifetime. Please note that Industry endurance perceptions have improved from a very high (> 2%) failure rate expectation to a very low actual PPM level failure rate in the past few years.

FIGURE 5: CYCLING MODE EFFECT ON ENDURANCE TESTING



© 1997 Microchip Technology Inc.

EXAMPLE #1 - AUTOMOBILE ELECTRONIC COMPASS

A 16-byte (128-bits) segment from a 2K array stores data every time the power is turned off in an automobile electronic compass. The design engineers expect the system to power down an average of 5 times/day over a 10-year life in an 85°C case temperature environment. The projected Microchip cumulative failure rate under these conditions at the 10 year point is less than 8 PPM or 0.0008% failures in 10 years. The cumulative PPM failure rate graph is referenced on Figure 6.

EXAMPLE #2 FULL-FEATURED TELEPHONE

An 8-byte (64-bit) segment is used to store the last number redial on a stationary full feature phone operating in a room temperature environment. A 12K bit (12,288-bits) segment is used for storage of speed dial numbers on the same phone. This application has two major functions and therefore it will have two separate failure rate calculations.

The last number redial function is expected to be utilized an average of 20 times/day over a 10 year life. Each speed dial is expected to be updated an average of 0.1 times/day over a 10 year life.

Figure 7, the cumulative failure rate graph for these two conditions indicates an extremely low failure rate in the projected 10 year lifetime. The failure rate begins beyond 20 years as shown on the attached cumulative PPM failure rate graph.

EXAMPLE #3 - LASER PRINTER

A serial EEPROM could have many functions in a laser printer. A function that would likely require the most cycles/day is the maintenance log storage of the pages printed (estimated to be 100 cycles/day). Three bytes are utilized to store this number. The case temperature environment is estimated to be between 55°C and 85°C

The high number of cycles at an extreme temperature of 85°C indicate a failure rate of less than 1200 PPM through the first 5 years and 2600 PPM through the first 10 years.

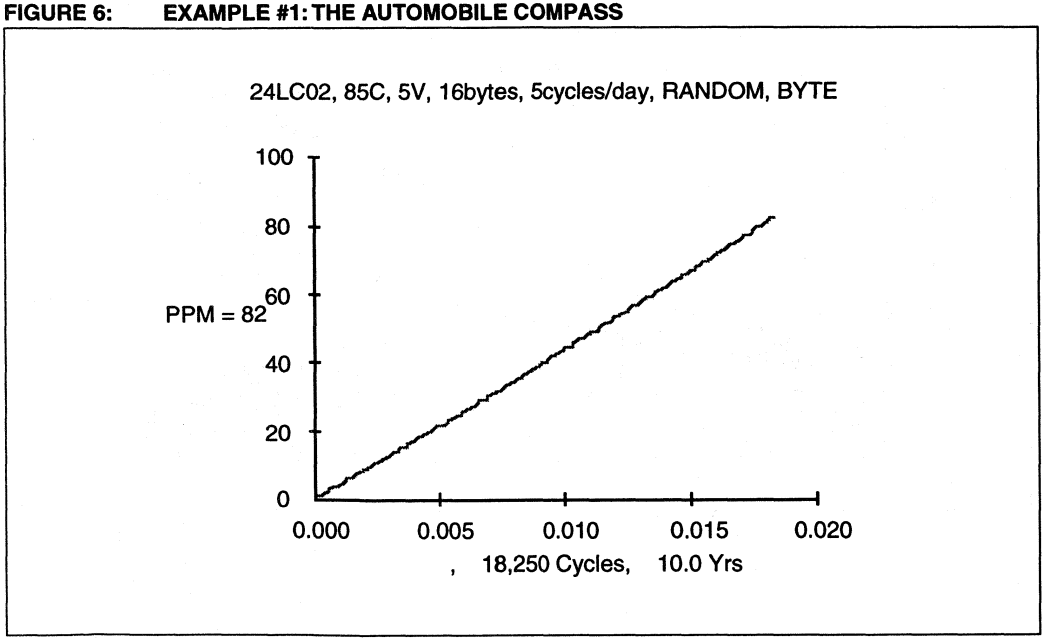
This failure rate can be dramatically reduced if the operating temperature is reduced to 55°C. The same 5 and 10 year PPM levels are reduced to 450 PPM and 1600 PPM at 55°C.

These failure rates are illustrated on Figure 8.

SUMMARY

Microchip Technology Inc. has recognized that increasing reliability in serial EEPROMs through increased endurance is not a function of extending the life of the intrinsic erase/write cycle, but depends on reducing defect-density failure rates. Design engineers characterizing EEPROM memory needs for an application and evaluating EEPROM components from various manufacturers need to understand not only the relationship between the application and expected use and failure mechanisms, but also how the manufacturer has arrived at published endurance data for its components.

FIGURE 6: EXAMPLE #1: THE AUTOMOBILE COMPASS



© 1997 Microchip Technology Inc.

Microchip has developed a program to calculate the cumulative PPM failure rates for specific system conditions, as shown on the previous pages. This program is being transferred onto menu driven DOS floppy disks. These disks will be available in December 1992 to the system designers, reliability engineers, and component engineers to project endurance failure rates for specific system conditions. These disks will be periodically updated to reflect Microchip's most recent endurance data.

FIGURE 7: EXAMPLE #2: THE FULL FEATURED PHONE

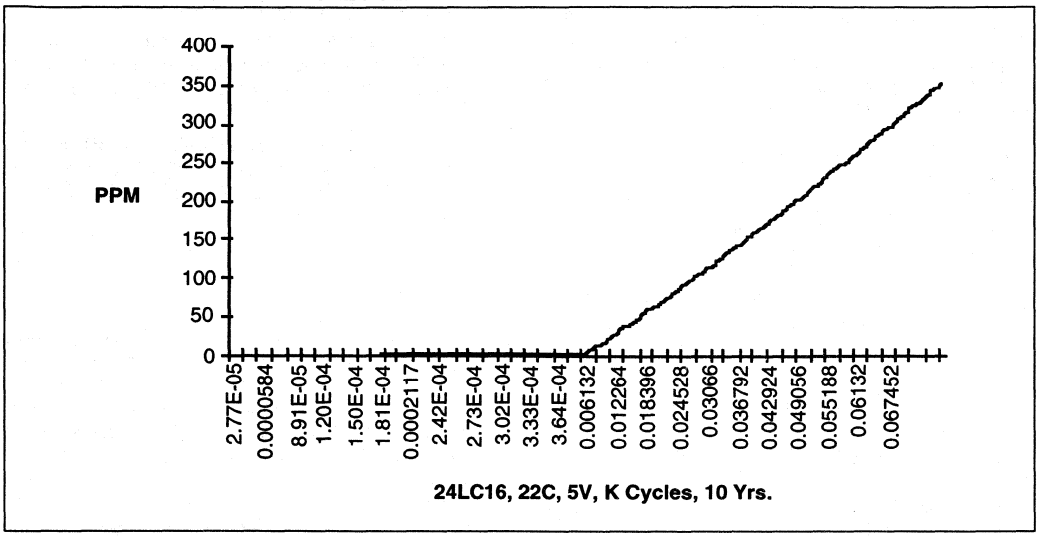
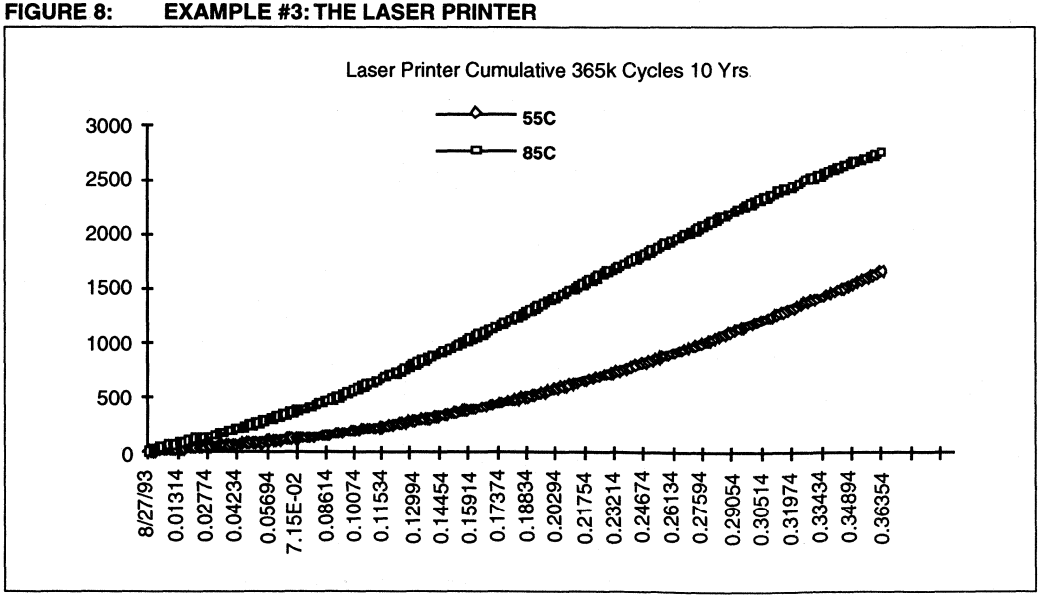


FIGURE 8: EXAMPLE #3: THE LASER PRINTER



© 1997 Microchip Technology Inc.

1.8 Volt Technology – Benefits

Author: *Tom Tyson*
Microchip Technology Inc.

Portability is and will continue to be one of the key features driving the design and development of new electronic equipment. Existing products such as palmtop computers, cellular phones and data acquisition devices are becoming increasingly smaller and more powerful. Handheld markets will demand that product feature sets be enhanced and that battery operating life be increased. These market requirements, in turn, will put more demands on battery and integrated circuit technology.

In order to optimize battery technology ICs must be able to operate at voltages well below the 5V range that most ICs operate at today. In fact, even integrated circuits operating at 3 volts do not make the most of the available battery technology. Thus the need for devices that operate below 3 volts.

BATTERY TECHNOLOGY OVERVIEW

Primary versus Secondary

In portable or handheld equipment there are two major battery types employed; primary and secondary. Primary cells are disposable, and alkaline and lithium cells are the main primary batteries in use today. Alkaline batteries are extremely common in consumer applications such as electronic games, cordless phones and palmtop computers. While lithium batteries are used as a primary power source in other consumer products such as cameras and as a back-up supply in palmtops.

Secondary cells are rechargeable batteries. Among the many secondary cells available, lead-acid, Ni-cad and Nickel Metal Hydride (NMH) are the most popular.

Voltage Ratings

Batteries for portable applications, whether primary or secondary, have three different voltage ratings; Rated or nominal, Operating, and End-of-life. The rated voltage is usually the open circuit voltage, which in the case of alkaline "AA" cells is 1.5V. Under normal load the "operating voltage" is realized and in all cases is less than the nominal voltage. The final battery voltage rating is the end-of-life voltage, which is defined as the voltage at which 100% of the usable power of the battery is consumed, or as 75% of the operating voltage. In the case of the same alkaline battery the end-of-life voltage is 0.9 volts. Table 1 lists the operating and end-of-life voltages for a single "AA" or equivalent cell of the five battery types mentioned above.

TABLE 1: BATTERY VOLTAGE SUMMARY

Battery Type	Operating Voltage	End-of-Life Voltage
ALKALINE	1.2V	0.9V
LITHIUM	2.7V	2.0V
LEAD-ACID	2.0V	1.75V
NI-CAD	1.2V	0.9V
NMH	1.2V	1.0V

With applications employing primary cells this means that batteries are replaced less often, thus reducing operating cost as well as having a positive effect on the environment due to the use and disposal of fewer batteries. In systems with secondary cells time between recharging increased, thus enhancing the performance of the product.

LOW VOLTAGE APPLICATIONS

As mentioned previously, the need for increased portability will drive the employment of low voltage technology. Portable and handheld applications requiring low voltage (sub 3V) technology will be numerous and will in many cases be high volume. These include a variety of personal communications devices such as cellular and cordless phones, computing devices like palmtops and portable PCs, and a number of data acquisition devices for a variety of medical, industrial and commercial applications.

© 1997 Microchip Technology Inc.

MICROCHIP'S LOW VOLTAGE PRODUCT POSITION

Prior to the announcement of a family of 1.8V Serial EEPROMs in November of 1992, Microchip had already established itself as a leader in low voltage semiconductor products. Microchip's current product portfolio consists of: (1) Serial EEPROMs that operate, both READ and WRITE, down to the following voltage levels: 1.8V, 2.0V, 2.5V and 5V; (2) ROM based microcontrollers that operate down to 2V; and (3) 3V One Time Programmable EPROMs.

The 1.8V Serial EEPROM family operates down to 1.8V without the relaxation of any specification. This includes all AC, DC retention and endurance parameters. Currently the 1.8V family consists of three 3-wire devices, in densities ranging from 1K to 4K bits. Packaging options include 8 pin SOIC and 8 pin PDIP devices. Typical operating currents for these devices, at 1.8V, are in the 70uA range, resulting in power consumption levels of less than 150uW. This compares to 1 to 5 mW for comparable devices operating at 5V. In the near future, the 1.8V family of Serial EEPROMs will be expanded to include all 2-wire devices.

SUMMARY

With the introduction of 1.8V Serial EEPROMs, Microchip has made available silicon technology that optimizes existing battery technology. This over time will result in the development, production, marketing and application of more portable and higher performance handheld and portable computing, telecommunications, and data acquisition products.

© 1997 Microchip Technology Inc.

MICROCHIP

AN551

Serial EEPROM Solutions vs. Parallel Solutions

Author: *Tom Tyson*
Microchip Technology Inc.

In searching for solutions to their system non-volatile memory requirements, equipment, systems and product designers are faced with a plethora of design related issues and trade-offs. The non-volatile memory options available to them offer a variety of different device features including performance, ease of design, power consumption, operating voltage, programmability, density, and physical size. For the most part these non-volatile memory options can be grouped into two major categories: serial solutions and parallel solutions. This paper discusses the attributes of each, conducts a comparative analysis and in the process identifies the benefits and advantages of Serial EEPROMs.

SERIAL EEPROMS

The main feature that makes a device a "Serial" and sets it apart from parallel devices is, as its name implies, the ability to communicate through a serial interface. This ability has numerous benefits. First, serial communication is accomplished with a minimum number of I/O's. Serial EEPROMs require only two to four lines (depending on the hardware and software protocol) for complete communication; memory addressing, data input and output, and device control. Thus, the hardware interface requirements for Serial EEPROMs are kept at a minimum. The most common Serial EEPROMs in use today are devices that utilize a 2-wire protocol.

Another benefit of serial communication is package size. Ranging from densities of 256 to 16K bits, most Serial EEPROMs today are available in space-saving 8 pin PDIP and 150mil wide SOIC packaging. This obviously is very beneficial for applications where product size and weight is a key design factor. The final benefit is low current consumption. Due to a limited number of I/O ports and therefore on-chip support requirements, operating currents for Serial EEPROMs are usually well below 3 milliamperes.

Other features of Serial EEPROMs include:

1. Byte programmability—The ability to erase and program one byte at a time without affecting the contents of the other memory locations in the array.
2. Clock rates of up to 6MHz—2-wire devices are rated at 100KHz and 400KHz per the standard I^2C protocol, while 3-wire devices can be operated at 6M Hz rates.
3. Low voltage operation—Microchip has introduced a family of devices that operate, both read and write, down to 1.8V. This family complements other 2V and 2.5V low voltage Serial EEPROM families available.

PARALLEL NON-VOLATILE MEMORIES

There are a number of memory devices that fall into this category. The major ones include Parallel EEPROMs, Flash memory products, EPROMs, and SRAMs with battery back-up.

The main common feature of all of these devices is that communication with the device is done through a parallel interface, which results in a high system clock rate. Each type of device has separate data, address and control lines. Thus pin counts are in the 24 to 40 pin range. This also results in relatively large and costly packages and large footprints, even with the most advanced surface mount packages like TSOP. SRAMs with on-board batteries require DIP package heights that are significantly higher than those of standard DIP packages, adding to its package size and cost disadvantage.

Parallel EEPROM and battery backed-up SRAMs are the only two of the four major types of parallel non-volatile memories that have the capability to erase and program one byte at a time. EPROM and Flash devices require the whole array or at least large sectors to be erased prior to reprogramming.

SERIAL VERSUS PARALLEL

Serial EEPROMs have five major advantages over parallel non-volatile memories.

1. Lower Current Consumption - The maximum operating current (at 5 volts operating voltage) of a 16K serial EEPROM device is approximately an order of magnitude less than that of an equivalent density parallel EEPROM. Operating currents for 16K Serials are specified at 3mA, while 16K parallel devices are specified at 30mA and above. This relationship will continue as 64K serial devices are introduced. Since power consumption is directly proportional to current consumption, the lower the current the lower the power consumption.

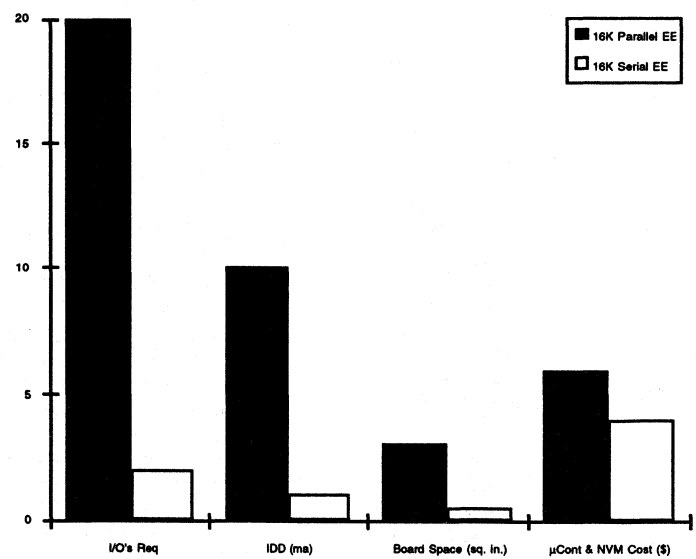
© 1997 Microchip Technology Inc.

2. Lower Voltage - Serial EEPROMs have been available in single supply low voltage options for some time. As mentioned above, Microchip has low voltage Serial EEPROMs that operate down to 1.8V, as well as other low voltage Serials that function down to 2.0V or 2.5V volts. 3V EPROMs and parallel EEPROMs and single voltage 5V flash devices are just being introduced to the market. (Most flash devices on the market today require 12V for programming in addition to the 5 volts required for normal operation). Low voltage operation also has a positive effect on power consumption. A reduction in the operating voltage from 5 volts to 1.8 volts will result in a power consumption reduction of almost 90% and almost a 65% reduction in power if the operating voltage is reduced from 3V to 1.8V.
3. Programmability - Neither currently available Flash devices nor EPROMs have the ability to program one byte at a time. Erasing is an array or sector function. Therefore, whenever one byte needs to be reprogrammed the entire array or sector must be reprogrammed. This procedure takes a relatively long amount of time to complete, time which may not be available, as is the case when storing critical parameters or data during inadvertent and unexpected system power loss. This procedure also requires software overhead to manage the retrieval and reprogramming operation.
4. Physical Size - Again, when comparing a 16K Serial EEPROM to a 16K parallel device the serial has a significant advantage. The area of the 150mil 8 pin SOIC footprint is less than 50K square mils. This compares to an area of more than 250K square mils for a 24 pin SOIC and almost 800K square mils for a 24 pin 500mil DIP package footprint.
5. I/O Requirements - Serials only require 2 to 4 input or output lines for complete communications. Most parallel devices require at least 22 lines, depending on the memory density. This results in increased microcontroller/microprocessor overhead and additional real estate to accommodate the numerous hardware lines.

The advantages that parallel devices currently have over serial EEPROMs is memory density and AC performance. However, in most microcontroller based applications for which Serial EEPROMs are intended, high density and AC are not the most critical design issues or most needed product features.

The key benefits of Serial EEPROM solutions as a result of the advantages outlined above, are reduced system costs, enhanced feature sets, and improved system performance. System size and weight is reduced and power sourcing requirements are kept at a minimum. The following graph compares some of the main attributes of a 16K Serial EEPROM device to a 16K Parallel device.

16K Serial vs. 16K Parallel Benefits



© 1997 Microchip Technology Inc.

USES AND APPLICATIONS OF SERIAL EEPROMS

Uses of Serial EEPROMs

The days of simply being a DIP switch replacement for Serial EEPROMs is over. Here is a list of the functions that Serial EEPROMs perform in a variety of computer, industrial, telecommunication, automotive and consumer applications:

1. Memory storage of channel selectors or analog controls (volume, tone, etc.)
2. Power down storage and retrieval of events such as fault detection or error diagnostics
3. Electronic real time event or maintenance log such as page counting
4. Configuration storage
5. Last number redial and speed dial storage
6. User in-circuit look-up tables

Serial EEPROM Applications

Serial EEPROMs have found homes in hundreds of embedded control applications in all major application markets. The following list demonstrates the number and variety of applications for serial EEPROMs.

Market	Applications
CONSUMER	TV tuners, VCRs, CD players, cameras, radios, and remote controls
COMPUTER/OA	Printers, copiers, PCs, palmtop and portable computers, disk drives and organizers
INDUSTRIAL	Bar code readers, point-of-sale terminals, smart cards, lock boxes, garage door openers, test measurement equipment and medical equipment
TELECOMM	Cellular, cordless and full feature phones, faxes, modems, pagers, and satellite receivers
AUTOMOTIVE	Air bags, anti-lock brakes, odometers, radios and keyless entry

Using Serial EEPROMs for critical data and configuration storage has only recently become a reality. The current offerings of 2- and 3-wire serial devices offers the systems designer interesting alternatives to the standard parallel EEPROM devices. The Serial EEPROM is basically a standard EEPROM array without the normal parallel data and address I/O. These functions are handled via serial I/O ports coupled with internal self-timed state machines. Not only will the serial device save power, board space, and cost, but they also offer the advantage of fewer I/O and consequently power in the embedded microcontroller because less I/O are needed to control the same functions. A typical embedded application is shown in Figure 1-A, depicting a controller and several functions used in a personal communications device, such as a mobile or portable phone. The EEPROM stores speed dial and last number redial numbers, credit card numbers, ID numbers, and configuration parameters.

Figure 1-B shows these same functions using a controller with fewer I/O and a Serial EEPROM. There is no loss of functionality but a significant savings in current, board space, I/O pads, and cost. The serial solution employs 8 to 16 less I/O on the microcontroller, freeing up much needed functionality, and possibly allowing for a much smaller device package and downsized circuit boards.

7

FIGURE 1-A AND -B: TYPICAL EMBEDDED APPLICATION VS. ONE WITH FEWER I/O AND A SERIAL EPROM

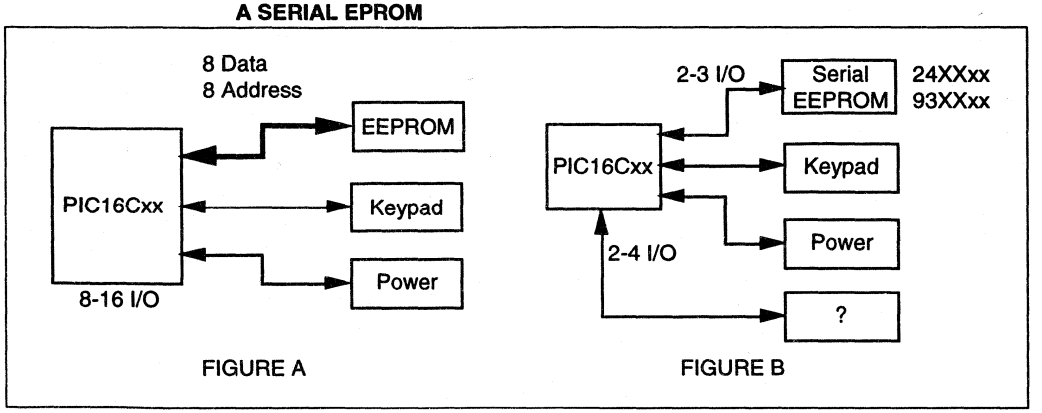


FIGURE A FIGURE B

SUMMARY

Serial EEPROMs are ideal cost effective solutions to all non-volatile memory embedded control applications that require: 1) A small footprint space saving format; 2) The ability and ease of programming one byte at a time; 3) Low current consumption and low operating voltage; 4) Low microcontroller overhead and support; and, 5) The best price performance non-volatile memory solution available.

Their size, ease of programmability, low power consumption, and low cost make Serial EEPROMs extremely suitable for all the fast growing handheld and portable battery powered computer, personal communications, medical and industrial markets.

© 1997 Microchip Technology Inc.

MICROCHIP

AN558

Using the 24xx65 and 25xx32 with Stand-alone PIC16C54 Code

Authors: Dick Fisher and Bruce Negley
Microchip Technology Inc.

USING THE SMART SERIAL™ SERIES

With the advent of CMOS silicon devices came the battery powered application. The battery powered application required more functionality and thus more power from the microcontroller which required more power from the batteries which cost more. Just as nature has a definite cycle, so it seems, does the portable application. Nowhere has this been more evident than in the areas of personal communication and data acquisition. Never before in the history of the industrial revolution have the type of applications seen emerging today been possible. Every man-made object that requires human or machine interface, has the potential to be controlled by embedded circuits and to be powered by the advanced technology batteries available today. Along with this increased functionality comes the requirement for MORE MEMORY. More in a smaller package, at a better price requiring less power. The majority of the hand-held embedded applications either use what little non-volatile memory that is available on the controllers or they use external devices. Parallel EEPROMs require too many I/O connects, which is the major source of device active current. Serial EEPROMs are typically the answer for these applications and the industry standard I²CTM is the industry leader (>70% worldwide). The I²C standard specifies a maximum 16K bit address space. How does the personal communications system designer, who requires more memory, solve this problem?

THE I²C DILEMMA

The I²C serial bus has many advantages over other common serial interfaces for serial embedded devices. The I²C bus with level-triggered inputs offers better noise immunity over edge triggered technology. Opcodes are not needed to communicate with storage devices because all interfaces are intuitive and comparable to parallel devices. The I²C protocol assigns a slave address for each unique device or device family. Microchip Technology, and most non-volatile suppliers, use the slave address of 1010 for serial electrical erasable programmable read only memory (Serial EEPROM). The protocol also facilitates up to a maximum of 16K bytes of memory on the bus via the 8-bit address and the three device or memory block select pins A0, A1, and A2. Herein lies the dilemma: with the advent of the more sophisticated personal communication devices such as cellular and full featured phones, personal digital assistants and palm-top computers, 16K bytes is not enough!

Smart Serial Products

The Smart Serial concept grew from the industry need for increased memory requirements in I²C embedded applications, smarter endurance performance, security needs, and the need for more functionality at lower power demands. Currently, Microchip Technology Inc. has the 24XX65 and 24xx32 devices available. All comments in this application note pertain primarily to the 24XX65 but the routines and general architectural comments apply to the 24xx32 as well.

The user should reference the individual data sheets for specific differences.

The Microchip 24xx65 is a serial memory device with 64K bits (8Kx8) capacity and additional patent pending unique features not found any where else. First let's look at the current I²C addressing scheme, the cascadable solution, and finally the Microchip total embedded systems solution.

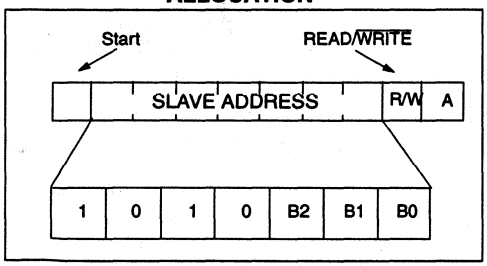
I²C Addressing

The I²C protocol utilizes a master/slave bi-directional communication bus. The master, usually a microcontroller, which controls the bus, generates the serial clock (SCL) and originates the start and stop conditions. A Serial EEPROM is considered a slave device and is defined as a transmitter during read operations and generates acknowledges when receiving data from the master. The start and stop bits are utilized to control the bus. Normal operation begins with a start bit and ends with a stop bit. Following a start, commands begin with an 8 bit 'control' byte originated by the master. The control byte identifies the slave device to be addressed and defines the operation to take place. A typical control byte for a Serial EEPROM (slave address = 1010) is shown in Figure 1. The control byte, therefore, consists of a start bit, a four-bit slave address, a read/write bit and an acknowledge. The slave address consists of the 1010 identifying address plus the three block or chip select bits.

7

© 1997 Microchip Technology Inc.

FIGURE 1: CONTROL BYTE ALLOCATION



All memory and peripheral devices on the I^2C bus conform to this sequence for identification and selection. There are 128 assigned slave addresses in the standard protocol. There is a 10-bit extension to the protocol for 1024 additional slave addresses.

THE CASCADABLE SOLUTION

Cascading is an addressing scheme used in the 24164 (2Kx8 SEEPROM) to enable using more than the 16K limit set forth by the standard. In this method the A0, A1, and A2 pins are mapped into bits 2, 3, and 4 of the 8-bit slave address. This approach allows the system designer to use the non-standard 24164 to increase the total memory of a given memory subsystem. This solution is definitely workable, but does not offer the user the system design flexibility needed for the newer and more complex systems. Most system architects would prefer to have a linear address space for program and data memory. Programmers also find the linear solution more attractive. The overhead required for bank switching and chip selection usually requires additional overhead and hardware.

THE ULTIMATE SOLUTION BY MICROCHIP

Microchip Technology has designed an addressing scheme based on the standard I^2C protocol and device addresses but incorporating an additional address byte for enabling the designer to use up to 256K bits per device and add from 1 to 8 devices on the system bus. This flexibility allows for future memory expansion and more advanced features in a smaller, more cost effective, design. This enhanced addressing combined with the many advanced and patent pending features of the 24xx65 make the 24xx65 an exciting and innovative device. It is the first in a family of sophisticated Smart Serial EEPROMs from Microchip Technology Inc.

24xx65 Features

The 24xx65 has an advanced architecture with the following features:

- 15-bit, 2-byte address field (14 bit for the 24xx32)
- 4K-bit High Endurance Block - 1 Million E/W cycles typical (Fixed at the last 4k bit block in the array)
- Programmable write protect security features with up to a 15 blocks of 4K bits
- 8 byte by 8 line input write cache for I^2C Fast Mode, burst mode capability, and use as a capture buffer

24xx65 Addressing

For the first byte or control byte, the 24xx65 adheres to the I^2C protocol (reference Figure 2). This is the first byte received following the start condition from the master device. The control byte consists of a four-bit control code for the 24xx65 (this is assigned as 1010 binary for read and write operations). The next three bits of the control byte are the device select bits (A2, A1, A0). They are used by the master device to select which of the eight devices are to be accessed. These bits are in effect the three most significant bits of the word address. The last bit of the control byte defines the operation to be performed. When set to a "1" a read operation is selected, when set to a "0" a write operation is selected. The least significant 13 bits of the next 2 bytes define the address of the first byte within the 8K block. The most significant byte is transferred first. Following the start condition, the 24xx65 monitors the SDA bus, checking the device type identifier being transmitted, upon receiving a 1010 code and appropriate device select bits, the slave device outputs an acknowledge signal on the SDA line. Depending on the state of the R/W bit, the 24xx65 will select a read or write operation.

The addressing scheme uses the standard first byte slave address format of the I^2C standard with the Microchip Technology-assigned 1010 slave address. The internal bus controller scans the next byte for bit 7 to be asserted indicating that a security operation is to take place. This will be further explained later. The remainder of the byte is composed of the most significant address bits. The next byte is the least significant address byte. See Figure 2 for graphical representation of this sequence. After receiving a valid 2 byte address and a stop bit, the 24xx65 will process this address according to bit "0" of the control byte and either wait for data to be written, if in a write sequence, or present the requested data if in a read sequence.

© 1997 Microchip Technology Inc.

ADVANCED FEATURES

Programmable security

The 24xx65 has a sophisticated security mechanism by which selected blocks of memory may be write protected by the user. The write sequence includes a bit for enabling the security protection scheme, bit 7 of the first address byte. When this bit is set to a one, the first byte following the address during a write sequence defines the security block. This includes a pointer to the starting 4K block to be protected (the write address), a write/erase flag, and a non-zero four bit code for determining the number of 4K blocks to protect up to the maximum of 15 (60K). The 4K blocks must be contiguous. The high endurance block cannot be protected. In a normal application the security block or blocks would be set after all code or look-up table data has been finalized. THIS OPERATION CAN ONLY BE PERFORMED ONCE. (See Figures 2 and 3.)

Total Endurance™

When defining endurance, we need to look at a few common definitions and possible misconceptions. Endurance with respect to EEPROMs is defined in number of Erase/Write (E/W) Cycles and is the most common rating referred to when discussing or specifying endurance. E/W ratings are based on the environmental and operating conditions of voltage, temperature, cycling mode and rate, for each byte in the application, not on the number of opcodes or control byte commands, and is never based on any read functions whether they be a data read or configuration read. If a part is rated at 100K E/W cycles, then each individual byte can be erased and written 100,000 times. This is probably the most common misinterpretation made by system designers. Endurance is thus an interactive application-specific reliability parameter. It is not a typical data sheet specification, such as a parametric AC/DC specification with benchmark standards for measurement. Microchip has done extensive predictive laboratory studies on Microchip 2- and 3-wire Serial EEPROMs. Applying the predictive data from the 24LC04B, which has similar characteristics, to the 4K high endurance block and assuming the following:

- A five-year life for a personal communication device
- An expected E/W cycles of 10 times per day
- A last number redial function of 11 bytes

Operational specifications:

Device	24xx65
Voltage	5
Temperature	25 C
Bytes/Cycle	11
E/W/Day	10
App. Life (Yr.)	5
Cycling Mode	BYTE
Data Pattern	RANDOM

The 4K HE block with 1 M E/W cycles typical, in this application, should yield the following results:

FIT	1.0
PPM	6
Time	5.0
Write cycles	18,250

FIGURE 2: CONTROL BYTE ALLOCATION

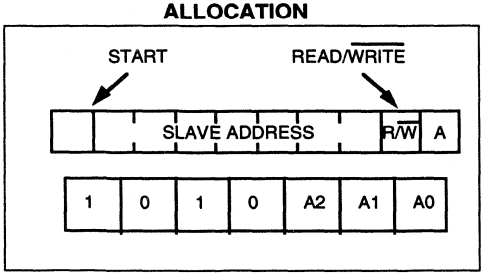
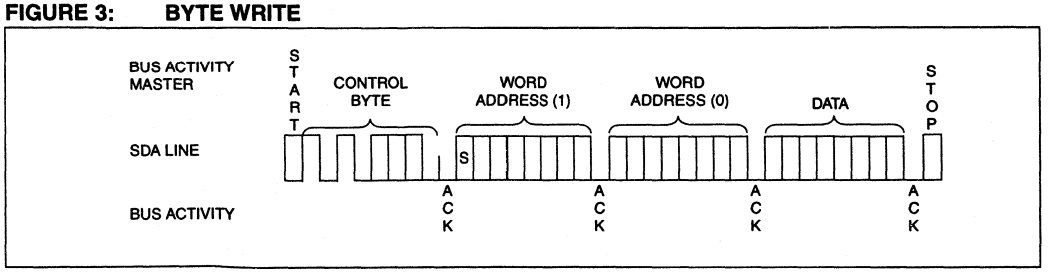
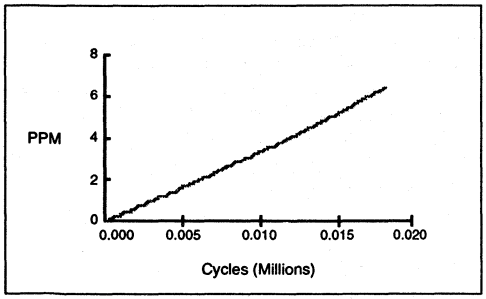


FIGURE 3: BYTE WRITE



© 1997 Microchip Technology Inc.

FIGURE 4: PPM RATE IN MILLIONS OF CYCLES



The results shown are predictive in nature and should reflect an accurate representation of the expected results. For a more detailed description of endurance see the related application notes AN537 and AN562 contained in this volume. (Figure 4.)

64 Byte Write Cache

The cache is arranged in 8-byte pages by 8 lines each. This yields a total of 64 bytes. The interface to the 24xx65 supports both the standard 100KHz mode and FAST mode at 400KHz. The input cache can therefore support a burst write option of up to 64 bytes. When using the I^2C protocol, an end of a page is defined by the transmission of a stop bit by the master. This sequence in the 24xx65 could be used to define pages from 8 to 64 bytes in length. The 8 byte by 8 line cache will roll over if a write is attempted past byte 8 of line 8, thus it can be used as a 64 byte capture or snapshot buffer. Each line is over written during a subsequent write to the same line. Faster memory and controller interfaces will become increasingly important in applications incorporating the ACCESS.bus™ interface standard.

Power Management

Increasingly, power management is becoming a predominant requirement for hand-held devices where a limited amount of power is available from the total power budget for a particular function. The 24xx65 has built-in power saving features such that the entire device is put into standby mode upon receiving a stop bit or an abort when in a read sequence, and after the completion of writing the data in the cache lines to the array when in an erase/write sequence. When the device is in standby mode, the only active circuit is the input circuit for the I^2C clock. This yields a standby current that is the current consumption of this lone input and the normal leakage current of the silicon and typically will be less than 2 μA.

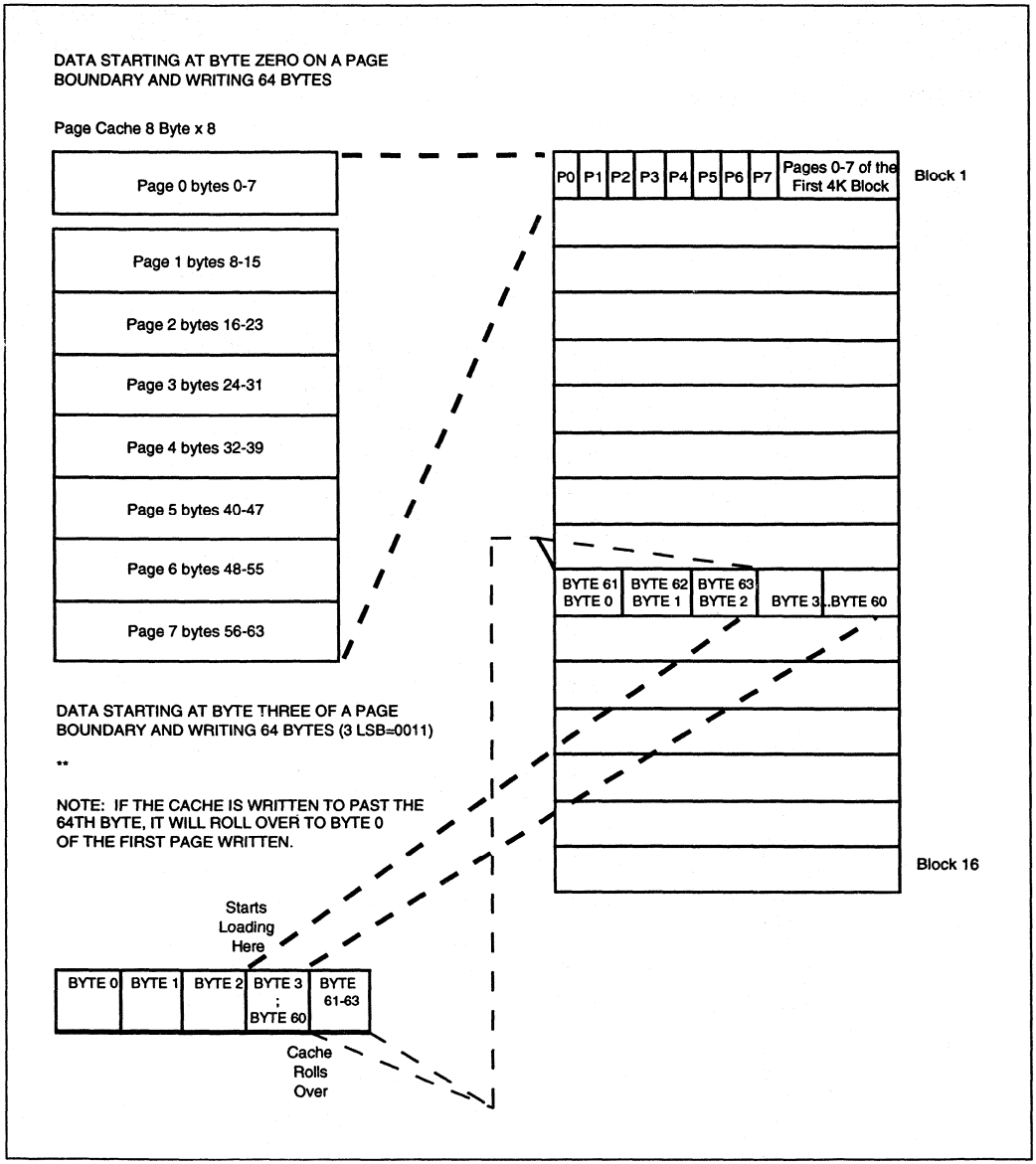
The 24xx65 is the first device available in the Smart Serial product line. The features and capabilities initiated with this device will become standard on all future Microchip Technology Inc. serials and some features will be enhanced, such as the security options.

Appendix A of this application note contains the required PIC16C54 assembly routines for setting the security function, addressing the Smart Serial devices and using the most common addressing, read, write, and data polling functions for the I^2C bus.

All of the list programs are complete stand-alone programs.

© 1997 Microchip Technology Inc.

FIGURE 5: CACHE WRITE



7

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: 64KBYTER.ASM

```
MPASM 01.40 Released         64KBYTER.ASM   1-16-1997  16:31:03         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54
                     00002 ;*****************************************************************
                     00003 ;       64K Byte Read Program (138 bytes)
                     00004 ;
                     00005 ;       This program demonstrates how to interface a
                     00006 ;       Microchip PIC 16C54 to a 24LC65 Serial E≤ device
                     00007 ;       and perform a random read operation. This program
                     00008 ;       will read 8 consecutive addresses in the 'random
                     00009 ;       read' mode.  This entails setting the address pointer
                     00010 ;       before doing the read command for each address.
                     00011 ;
                     00012 ;       Another, more efficient method of reading consecutive
                     00013 ;       addresses is the 'sequential read' mode.  This involves
                     00014 ;       sending the control byte and address for the first byte
                     00015 ;       to read, then continuing to provide clocks for the next
                     00016 ;       addresses.  The device will automatically increment the
                     00017 ;       address.  An example of the sequential read mode is
                     00018 ;       provided in the '64kseqr.asm' file.
                     00019 ;
                     00020 ;       Timing is based on using the PIC in 'XT' mode
                     00021 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00022 ;       will be approximately 60 Khz for this setup.
                     00023 ;
                     00024 ;       As an option, the user may connect a LED to pin 18
                     00025 ;       on the PIC (use about a 1K resistor in series) as an
                     00026 ;       acknowledge fail indicator.  This LED will come on if
                     00027 ;       the serial EE fails to acknowledge correctly.
                     00028 ;
                     00029 ;        PIC to Serial EE Connections:
                     00030 ;
                     00031 ;           PIC          Serial EE
                     00032 ;       ------------    ------------
                     00033 ;       Pin 12 (RB6) -->  SCLK
                     00034 ;       Pin 13 (RB7) -->  SDATA
                     00035 ;
                     00036 ;       PIN 18 (RA1) -->  Acknowledge fail LED  (Optional)
                     00037 ;
                     00038 ;
                     00039 ;       Program:          64KBYTER.ASM
                     00040 ;       Revision Date:
                     00041 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00042 ;
                     00043 ;
                     00044 ;*****************************************************************
                     00045 ;           Register Definitions
                     00046 ;*****************************************************************
00000005             00047 port_a  equ     5h      ; port 5 (port a) used for LED display
00000006             00048 port_b  equ     6h      ; port 6 (port b) used for data and
                     00049                         ; clock lines
0000000A             00050 eeprom  equ     0ah     ; bit buffer
0000000B             00051 bycnt   equ     0bh     ; byte counter for read mode
0000000C             00052 addr    equ     0ch     ; word 0 address counter
0000000D             00053 datai   equ     0dh     ; data input register
0000000E             00054 datao   equ     0eh     ; data output register
```


© 1997 Microchip Technology Inc.

```
  0000000F          00055 slave   equ    0fh     ; device address 1010xxx0)
  00000010          00056 txbuf   equ    10h     ; transmit buffer
  00000011          00057 count   equ    11h     ; bit counter
  00000012          00058 bcount  equ    12h     ; byte counter
  00000015          00059 loops   equ    15h     ; delay loop counter
  00000016          00060 loops2  equ    16h     ; delay loop counter
  00000017          00061 addr1   equ    17h     ; word 1 address counter
                    00062 ;*****************************************************************
                    00063 ;            Bit Definitions
                    00064 ;*****************************************************************
  00000007          00065 di      equ    7       ; eeprom input bit
  00000006          00066 do      equ    6       ; eeprom output bit
  00000007          00067 sdata   equ    7       ; serial EE data line (port_b,pin 13)
  00000006          00068 sclk    equ    6       ; serail EE clock line (port_b,pin 12)
  00000001          00069 ackf    equ    1       ; acknowledge fail LED (port_a)
                    00070 ;*****************************************************************
                    00071 ;
01FF                00072         org    01ffh   ; set reset vector
01FF 0A6B           00073         goto   PWRUP
0000                00074         org    000h    ;
0000 0A6B           00075         goto   PWRUP
                    00076 ;
                    00077 ;*****************************************************************
                    00078 ;       DELAY ROUTINE
                    00079 ;       This routine takes the value in 'loops'
                    00080 ;       and multiplies it times 1 millisecond to
                    00081 ;       determine delay time.
                    00082 ;*****************************************************************
0001                00083 WAIT
                    00084 ;
0001 0C6E           00085 top      movlw   .110           ; timing adjustment variable
0002 0036           00086          movwf   loops2
0003 0000           00087 top2     nop                    ; sit and wait
0004 0000           00088          nop
0005 0000           00089          nop
0006 0000           00090          nop
0007 0000           00091          nop
0008 0000           00092          nop
0009 02F6           00093          decfsz  loops2, F      ; inner loops complete?
000A 0A03           00094          goto    top2           ; no, go again
                    00095                                 ;
000B 02F5           00096          decfsz  loops, F       ; outer loops complete?
000C 0A01           00097          goto    top            ; no, go again
000D 0800           00098          retlw   0              ; yes, return from sub
                    00099 ;
                    00100 ;*****************************************************************
                    00101 ;       Start Bit Subroutine
                    00102 ;       this routine generates a start bit
                    00103 ;       (Low going data line while clock is high)
                    00104 ;*****************************************************************
                    00105 ;
000E                00106 BSTART
000E 05E6           00107          bsf     port_b,sdata   ; make sure data is high
000F 0C3F           00108          movlw   b'00111111'
0010 0006           00109          tris    port_b         ; set data and clock lines for output
0011 04C6           00110          bcf     port_b,sclk    ; make sure clock is low
0012 0000           00111          nop
0013 05C6           00112          bsf     port_b,sclk    ; set clock high
0014 0000           00113          nop
0015 0000           00114          nop
0016 0000           00115          nop
0017 0000           00116          nop
0018 0000           00117          nop
0019 04E6           00118          bcf     port_b,sdata   ; data line goes low during
                    00119                                 ; high clock for start bit
001A 0000           00120          nop
```


© 1997 Microchip Technology Inc.

```
001B 0000          00121          nop
001C 0000          00122          nop
001D 0000          00123          nop
001E 0000          00124          nop                    ; timing adjustment
001F 04C6          00125          bcf     port_b,sclk    ; start clock train
0020 0000          00126          nop
0021 0000          00127          nop
0022 0800          00128          retlw   0
                   00129          ;
                   00130 ;*****************************************************
                   00131 ;       Stop Bit Subroutine
                   00132 ;       This routine generates a stop bit
                   00133 ;       (High going data line while clock is high)
                   00134 ;*****************************************************
0023               00135 BSTOP
0023 04E6          00136          bcf     port_b,sdata   ; make sure data line is low
0024 0C3F          00137          movlw   b'00111111'    ;
0025 0006          00138          tris    port_b         ; set data/clock lines as outputs
0026 04E6          00139          bcf     port_b,sdata   ; make sure data line is low
0027 0000          00140          nop
0028 0000          00141          nop
0029 0000          00142          nop
002A 05C6          00143          bsf     port_b,sclk    ; set clock high
002B 0000          00144          nop
002C 0000          00145          nop
002D 0000          00146          nop
002E 05E6          00147          bsf     port_b,sdata   ; data goes high while clock high
                   00148                                 ; for stop bit
002F 0000          00149          nop
0030 0000          00150          nop
0031 04C6          00151          bcf     port_b,sclk    ; set clock low again
0032 0000          00152          nop
0033 0000          00153          nop
0034 0000          00154          nop
0035 0800          00155          retlw   0
                   00156 ;
                   00157 ;*****************************************************
                   00158 ;       BITOUT routine takes the bit of data in 'do' and
                   00159 ;       transmits it to the serial EE device
                   00160 ;*****************************************************
0036               00161 BITOUT
0036 0C3F          00162          movlw   b'00111111'    ; set data,clock as outputs
0037 0006          00163          tris    port_b
0038 07CA          00164          btfss   eeprom,do      ; check for state of data bit to xmit
0039 0A3C          00165          goto    bitlow         ; low? go set data line low
003A 05E6          00166          bsf     port_b,sdata   ; high? set data line high
003B 0A3D          00167          goto    clkout         ; go toggle the clock
                   00168
003C 04E6          00169 bitlow   bcf     port_b,sdata   ; output a low bit
003D 05C6          00170 clkout   bsf     port_b,sclk    ; set clock line high
003E 0000          00171          nop
003F 0000          00172          nop
0040 0000          00173          nop
0041 0000          00174          nop
0042 04C6          00175          bcf     port_b,sclk    ; return clock line low
0043 0800          00176          retlw   0
                   00177 ;
                   00178 ;*****************************************************
                   00179 ;       BITIN routine reads one bit of data from the
                   00180 ;       serial EE device and stores it in the bit 'di'
                   00181 ;*****************************************************
0044               00182 BITIN
0044 05EA          00183          bsf     eeprom,di      ; assume input bit is high
0045 0CBF          00184          movlw   b'10111111'    ; make sdata an input line
0046 0006          00185          tris    port_b
0047 05E6          00186          bsf     port_b,sdata   ; set sdata line for input
```


© 1997 Microchip Technology Inc.

```
0048 05C6          00187          bsf     port_b,sclk     ; set clock line high
0049 0000          00188          nop                     ; just sit here a sec
004A 0000          00189          nop
004B 0000          00190          nop
004C 0000          00191          nop
004D 0000          00192          nop                     ;
004E 07E6          00193          btfss   port_b,sdata    ; read the data bit
004F 04EA          00194          bcf     eeprom,di       ; input bit was low, set 'di' accordingly
0050 04C6          00195          bcf     port_b,sclk     ; set clock line low
                   00196                                  ;
0051 0800          00197          retlw   0               ;
                   00198 ;
                   00199 ;*****************************************************************
                   00200 ;       Transmit Data Subroutine
                   00201 ;       This routine takes the byte of data stored in the
                   00202 ;       'datao' register and transmits it to the serial EE device.
                   00203 ;       It will then send 1 more clock to the serial EE for the
                   00204 ;       acknowledge bit.  If the ack bit from the part was low
                   00205 ;       then the transmission was sucessful.  If it is high, then
                   00206 ;       the device did not send a proper ack bit and the ack
                   00207 ;       fail LED will be turned on.
                   00208 ;*****************************************************************
0052               00209 TX
0052 0C08          00210          movlw   .8
0053 0031          00211          movwf   count           ; set the #bits to 8
                   00212          ;
0054               00213 TXLP
0054 04CA          00214          bcf     eeprom,do       ; assume bit out is low
0055 06F0          00215          btfsc   txbuf,7         ; is bit out really low?
0056 05CA          00216          bsf     eeprom,do       ; no, set it high
0057 0936          00217          call    BITOUT          ; send the bit to serial EE
0058 0370          00218          rlf     txbuf, F        ; rotate txbuf left
0059 02F1          00219          decfsz  count, F        ; 8 bits done?
005A 0A54          00220          goto    TXLP            ; no - go again
005B 0944          00221          call    BITIN           ; read ack bit
005C 06EA          00222          btfsc   eeprom,di       ; check ack bit
005D 0525          00223          bsf     port_a,ackf     ; set acknowledge fail LED if the
                   00224                                  ; device did not pull data low
                   00225          ;
005E 0800          00226          retlw   0
                   00227 ;
                   00228 ;*****************************************************************
                   00229 ;       Receive data routine
                   00230 ;       This routine reads one byte of data from the part
                   00231 ;       into the 'datai' register.  It then sends a high
                   00232 ;       ack bit to indicate that no more data is to be read
                   00233 ;*****************************************************************
005F               00234 RX
005F 006D          00235          clrf    datai           ; clear input buffer
0060 0C08          00236          movlw   .8              ; set # bits to 8
0061 0031          00237          movwf   count
0062 036D          00238 RXLP     rlf     datai, F        ; rotate datai 1 bit left
0063 0944          00239          call    BITIN           ; read a bit
0064 06EA          00240          btfsc   eeprom,di
0065 050D          00241          bsf     datai,0         ; set bit 0 if necessary
0066 02F1          00242          decfsz  count, F        ; 8 bits done?
0067 0A62          00243          goto    RXLP            ; no, do another
0068 05CA          00244          bsf     eeprom,do       ; set ack bit = 1
0069 0936          00245          call    BITOUT          ; to finish transmission
006A 0800          00246          retlw   0
                   00247 ;
                   00248 ;*****************************************************************
                   00249 ;       Power up routine
                   00250 ;       This is the program entry point.  I/O line status is
                   00251 ;       set for port A here.
                   00252 ;*****************************************************************
```

7

```
006B                00253 PWRUP
006B 0C00           00254           movlw  b'00000000'
006C 0005           00255           tris   port_a          ; set port A as all output
006D 0065           00256           clrf   port_a          ; all output lines low
006E 0A6F           00257           goto   READ
                    00258 ;
                    00259 ;*****************************************************************
                    00260 ;       READ (read routine)
                    00261 ;       This routine reads 8 consecutive addresses of the
                    00262 ;       serial EE device starting at address 00 in the
                    00263 ;       random access mode (non-sequential read). Reading
                    00264 ;       the device using the random access mode
                    00265 ;       requires that the address pointer be set for every
                    00266 ;       byte before the read takes place. The address pointer
                    00267 ;       is set by sending a 'write mode' control byte with the
                    00268 ;       address of the location to read.
                    00269 ;*****************************************************************
006F                00270 READ
                    00271 ;
006F 0425           00272           bcf     port_a,ackf ; clear the ack fail LED if on
0070 0C08           00273           movlw   .8          ; set number of bytes to read as 8
0071 0032           00274           movwf   bcount      ;
0072 0077           00275           clrf    addr1       ; set starting high address byte to 00
0073 006C           00276           clrf    addr        ; set starting low address byte to 00
                    00277                               ;
0074 090E           00278 rbyte     call    BSTART      ; generate start bit
                    00279                               ;
                    00280                               ; now send the write control byte and
                    00281                               ; address to set the pointer
                    00282                               ;
0075 0CA0           00283           movlw   b'10100000' ; get slave address (write mode)
0076 0030           00284           movwf   txbuf       ; into transmit buffer
0077 0952           00285           call    TX          ; and send it
0078 0217           00286           movf    addr1,w     ; get word 1 address
0079 0030           00287           movwf   txbuf       ; into transmit buffer
007A 0952           00288           call    TX          ; and send it
007B 020C           00289           movf    addr,w      ; get word 0 address
007C 0030           00290           movwf   txbuf       ; into transmit buffer
007D 0952           00291           call    TX          ; and send it
                    00292                               ;
                    00293                               ; now read one byte from the part
                    00294                               ;
007E 090E           00295           call    BSTART      ; generate start bit
007F 0CA1           00296           movlw   b'10100001' ; get slave address and read mode
0080 0030           00297           movwf   txbuf       ; into transmit buffer
0081 0952           00298           call    TX          ; and transmit it
0082 095F           00299           call    RX          ; read 1 byte from serial EE
0083 0923           00300           call    BSTOP       ; send stop bit to end transmission
0084 02AC           00301           incf    addr, F     ; add 1 to address counter
0085 02F2           00302           decfsz  bcount, F   ; yes, are all 8 bytes read?
0086 0A74           00303           goto    rbyte       ; no, do another byte
                    00304
0087 0CFF           00305           movlw   .255        ; long delay for scope
0088 0035           00306           movwf   loops       ; trigger purposes only
0089 0901           00307           call    WAIT
008A 0A6F           00308           goto    READ        ; yes, start all over
                    00309 ;
                    00310     END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXX----- ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.
```


© 1997 Microchip Technology Inc.

```
Program Memory Words Used:   140
Program Memory Words Free:   372
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: 64KBYTEW.ASM

```
MPASM 01.40 Released          64KBYTEW.ASM   1-16-1997  16:31:26          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54
                     00002 ;*****************************************************************
                     00003 ;       64K Byte Write Program (127 bytes)
                     00004 ;
                     00005 ;       This program demonstrates how to interface a
                     00006 ;       Microchip PIC 16C54 to the 24LC65 Serial E≤ device
                     00007 ;       and perform a byte write operation on 8 consecutive
                     00008 ;       addresses.
                     00009 ;
                     00010 ;       After each byte is written, time must be given to the
                     00011 ;       device for it to complete the write cycle before
                     00012 ;       the next command can be sent.  The easiest solution
                     00013 ;       is to consult the data book for the maximum write
                     00014 ;       cycle time and just wait that long before the next
                     00015 ;       command is sent(10ms for this device).  This program
                     00016 ;       demonstrates that solution.
                     00017 ;
                     00018 ;       Another, more efficient method of determining when the
                     00019 ;       write cycle is complete is called 'data polling.'  This
                     00020 ;       method is demonstrated in the program "64kdpoll."
                     00021 ;
                     00022 ;       As an option, the user may connect a LED to pin 18
                     00023 ;       on the PIC (use about a 1K resistor in series) as an
                     00024 ;       acknowledge fail indicator.  This LED will come on if
                     00025 ;       the serial EE fails to acknowledge correctly.
                     00026 ;
                     00027 ;       Timing is based on using the PIC in 'XT' mode
                     00028 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00029 ;       will be approximately 60 Khz for this setup.
                     00030 ;
                     00031 ;        PIC to Serial EE Connections:
                     00032 ;
                     00033 ;           PIC          Serial EE
                     00034 ;       ------------    ------------
                     00035 ;       Pin 12 (RB6) -->  SCLK
                     00036 ;       Pin 13 (RB7) -->  SDATA
                     00037 ;
                     00038 ;       PIN 18 (RA1) -->  Acknowledge fail LED  (Optional)
                     00039 ;
                     00040 ;
                     00041 ;       Program:          64KBYTEW.ASM
                     00042 ;       Revision Date:
                     00043 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00044 ;
                     00045 ;*****************************************************************
                     00046 ;           Register Definitions
                     00047 ;*****************************************************************
00000005             00048 port_a  equ     5h      ; port 5 (port_a) used for LEDs
00000006             00049 port_b  equ     6h      ; port 6 (port b) used for data and
                     00050                         ; clock lines
0000000A             00051 eeprom  equ    0ah      ; bit buffer
0000000B             00052 bycnt   equ    0bh      ; byte counter for read mode
0000000C             00053 addr    equ    0ch      ; word 0 address counter
0000000D             00054 datai   equ    0dh      ; data input register
```


© 1997 Microchip Technology Inc.

```
  0000000E          00055 datao   equ     0eh     ; data output register
  0000000F          00056 slave   equ     0fh     ; device address (1010xxx0)
  00000010          00057 txbuf   equ     10h     ; transmit buffer
  00000011          00058 count   equ     11h     ; bit counter
  00000012          00059 bcount  equ     12h     ; byte counter
  00000015          00060 loops   equ     15h     ; delay loop counter
  00000016          00061 loops2  equ     16h     ; delay loop counter
  00000017          00062 addr1   equ     17h     ; word 1 address counter
                    00063 ;*****************************************************************
                    00064 ;           Bit Definitions
                    00065 ;*****************************************************************
  00000007          00066 di      equ     7       ; eeprom input bit
  00000006          00067 do      equ     6       ; eeprom output bit
  00000007          00068 sdata   equ     7       ; serial EE data line (port_b,pin 13)
  00000006          00069 sclk    equ     6       ; serail EE clock line (port_b,pin 12)
  00000001          00070 ackf    equ     1       ; acknowledge fail LED (port_a,pin 18)
                    00071 ;*****************************************************************
                    00072 ;
                    00073 ;
01FF                00074         org     01ffh   ; set reset vector
01FF 0A5E           00075         goto    PWRUP
0000                00076         org     000h    ;
0000 0A5E           00077         goto    PWRUP
                    00078 ;
                    00079 ;*****************************************************************
                    00080 ;       DELAY ROUTINE
                    00081 ;       This routine takes the value in 'loops'
                    00082 ;       and multiplies it times 1 millisecond to
                    00083 ;       determine delay time.
                    00084 ;*****************************************************************
0001                00085 WAIT
                    00086 ;
0001 0C6E           00087 top       movlw   .110            ; timing adjustment variable
0002 0036           00088           movwf   loops2
0003 0000           00089 top2      nop                     ; sit and wait
0004 0000           00090           nop
0005 0000           00091           nop
0006 0000           00092           nop
0007 0000           00093           nop
0008 0000           00094           nop
0009 02F6           00095           decfsz  loops2, F       ; inner loops complete?
000A 0A03           00096           goto    top2            ; no, go again
                    00097                                   ;
000B 02F5           00098           decfsz  loops, F        ; outer loops complete?
000C 0A01           00099           goto    top             ; no, go again
000D 0800           00100           retlw   0               ; yes, return from sub
                    00101 ;
                    00102 ;*****************************************************************
                    00103 ;       Start Bit Subroutine
                    00104 ;       this routine generates a start bit
                    00105 ;       (Low going data line while clock is high)
                    00106 ;*****************************************************************
                    00107 ;
000E                00108 BSTART
000E 05E6           00109           bsf     port_b,sdata    ; make sure data is high
000F 0C3F           00110           movlw   b'00111111'
0010 0006           00111           tris    port_b          ; set data and clock lines for output
0011 04C6           00112           bcf     port_b,sclk     ; make sure clock is low
0012 0000           00113           nop
0013 05C6           00114           bsf     port_b,sclk     ; set clock high
0014 0000           00115           nop
0015 0000           00116           nop
0016 0000           00117           nop
0017 0000           00118           nop
0018 0000           00119           nop
0019 04E6           00120           bcf     port_b,sdata    ; data line goes low during
```


© 1997 Microchip Technology Inc.

```
                    00121                                   ; high clock for start bit
001A 0000           00122           nop
001B 0000           00123           nop
001C 0000           00124           nop
001D 0000           00125           nop
001E 0000           00126           nop                     ; timing adjustment
001F 04C6           00127           bcf     port_b,sclk     ; start clock train
0020 0000           00128           nop
0021 0000           00129           nop
0022 0800           00130           retlw   0
                    00131 ;
                    00132 ;*****************************************************
                    00133 ;       Stop Bit Subroutine
                    00134 ;       This routine generates a stop bit
                    00135 ;       (High going data line while clock is high)
                    00136 ;*****************************************************
0023                00137 BSTOP
0023 0C3F           00138           movlw   b'00111111'     ;
0024 0006           00139           tris    port_b          ; set data/clock lines as outputs
0025 04E6           00140           bcf     port_b,sdata    ; make sure data line is low
0026 0000           00141           nop
0027 0000           00142           nop
0028 0000           00143           nop
0029 05C6           00144           bsf     port_b,sclk     ; set clock high
002A 0000           00145           nop
002B 0000           00146           nop
002C 0000           00147           nop
002D 05E6           00148           bsf     port_b,sdata    ; data goes high while clock high
                    00149                                   ; for stop bit
002E 0000           00150           nop
002F 0000           00151           nop
0030 04C6           00152           bcf     port_b,sclk     ; set clock low again
0031 0000           00153           nop
0032 0000           00154           nop
0033 0000           00155           nop
0034 0800           00156           retlw   0
                    00157 ;
                    00158 ;*****************************************************
                    00159 ;       BITOUT routine takes one bit of data in 'do' and
                    00160 ;       transmits it to the serial EE device
                    00161 ;*****************************************************
0035                00162 BITOUT
0035 0C3F           00163           movlw   b'00111111'     ; set data,clock as outputs
0036 0006           00164           tris    port_b
0037 07CA           00165           btfss   eeprom,do       ; check for state of data bit to xmit
0038 0A3B           00166           goto    bitlow          ;
0039 05E6           00167           bsf     port_b,sdata    ; set data line high
003A 0A3C           00168           goto    clkout          ; go toggle the clock
                    00169
003B 04E6           00170 bitlow    bcf     port_b,sdata    ; output a low bit
003C 05C6           00171 clkout    bsf     port_b,sclk     ; set clock line high
003D 0000           00172           nop
003E 0000           00173           nop
003F 0000           00174           nop
0040 0000           00175           nop
0041 04C6           00176           bcf     port_b,sclk     ; return clock line low
0042 0800           00177           retlw   0
                    00178 ;
                    00179 ;*****************************************************
                    00180 ;       BITIN routine reads one bit of data from the
                    00181 ;       serial EE device and stores it in 'di'
                    00182 ;*****************************************************
0043                00183 BITIN
0043 05EA           00184           bsf     eeprom,di       ; assume input bit is high
0044 0CBF           00185           movlw   b'10111111'     ; make sdata an input line
0045 0006           00186           tris    port_b
```


© 1997 Microchip Technology Inc.

```
0046 05E6          00187          bsf     port_b,sdata    ; set sdata line for input
0047 05C6          00188          bsf     port_b,sclk     ; set clock line high
0048 0000          00189          nop                     ; just sit here a sec
0049 0000          00190          nop
004A 0000          00191          nop
004B 0000          00192          nop
004C 0000          00193          nop                     ;
004D 07E6          00194          btfss   port_b,sdata    ; read the data bit
004E 04EA          00195          bcf     eeprom,di       ; input bit was low
004F 04C6          00196          bcf     port_b,sclk     ; set clock line low
                   00197                                  ;
0050 0800          00198          retlw   0               ;
                   00199 ;
                   00200 ;*****************************************************************
                   00201 ;       Transmit Data Subroutine
                   00202 ;       This routine takes the byte of data stored in the
                   00203 ;       'datao' register and transmits it to the serial EE device.
                   00204 ;       It will then send 1 more clock to the serial EE for the
                   00205 ;       acknowledge bit.  If the ack bit from the part was low
                   00206 ;       then the transmission was sucessful.  If it is high, then
                   00207 ;       the device did not send a proper ack bit and the ack
                   00208 ;       fail LED will be turned on.
                   00209 ;*****************************************************************
0051               00210 TX
0051 0C08          00211          movlw   .8
0052 0031          00212          movwf   count           ; set the #bits to 8
                   00213          ;
0053               00214 TXLP
0053 04CA          00215          bcf     eeprom,do       ; assume bit out is low
0054 06F0          00216          btfsc   txbuf,7         ; is bit out really low?
0055 05CA          00217          bsf     eeprom,do       ; otherwise data bit =1
0056 0935          00218          call    BITOUT          ; serial data out
0057 0370          00219          rlf     txbuf, F        ; rotate txbuf left
0058 02F1          00220          decfsz  count, F        ; 8 bits done?
0059 0A53          00221          goto    TXLP            ; no - go again
005A 0943          00222          call    BITIN           ; read ack bit
005B 06EA          00223          btfsc   eeprom,di       ; check ack bit
005C 0525          00224          bsf     port_a,ackf     ; set acknowledge fail LED if the
                   00225          ;
005D 0800          00226          retlw   0
                   00227 ;
                   00228 ;*****************************************************************
                   00229 ;       Power up routine
                   00230 ;       This is the program entry point, which in this case simply
                   00231 ;       sets the port_a I/O lines and directs control to the
                   00232 ;       byte write routine.
                   00233 ;*****************************************************************
005E               00234 PWRUP
005E 0C00          00235           movlw  b'00000000'
005F 0005          00236           tris   port_a           ; set port A as all output
0060 0065          00237           clrf   port_a           ; all output lines low
0061 0A62          00238           goto   WRBYTE
                   00239
                   00240 ;*****************************************************************
                   00241 ;       Byte Write Routine
                   00242 ;       This routine writes the data in "datao" to
                   00243 ;       8 consecutive bytes in the serial EE device starting
                   00244 ;       at address 00.  This routine waits 10mS after every
                   00245 ;       byte to give the device time to do the write.  This
                   00246 ;       program repeats forever.
                   00247 ;*****************************************************************
                   00248 ;
0062               00249 WRBYTE
                   00250 ;
0062 0065          00251          clrf    port_a      ; clear all LEDs
0063 0CA0          00252          movlw   b'10100000' ; set slave address and write mode
```


© 1997 Microchip Technology Inc.

```
0064 002F           00253          movwf   slave
0065 0C55           00254          movlw   b'01010101' ; set data to write as 55h
0066 002E           00255          movwf   datao
                    00256 ;
0067 0C08           00257          movlw   .8          ; set number of bytes
0068 0032           00258          movwf   bcount      ; to write as to 8
0069 0077           00259          clrf    addr1       ; set high address byte to 00
006A 006C           00260          clrf    addr        ; set low address byte to 00
                    00261                              ;
006B 090E           00262 byte     call    BSTART      ; generate start bit
006C 020F           00263          movf    slave,w     ; move slave address
006D 0030           00264          movwf   txbuf       ; into transmit buffer
006E 0951           00265          call    TX          ; and send it
006F 0217           00266          movf    addr1,w     ; move word 1 address
0070 0030           00267          movwf   txbuf       ; into transmit buffer
0071 0951           00268          call    TX          ; and send it
0072 020C           00269          movf    addr,w      ; move word 0 address
0073 0030           00270          movwf   txbuf       ; into transmit buffer
0074 0951           00271          call    TX          ; and send it
0075 020E           00272          movf    datao,w     ; move data byte
0076 0030           00273          movwf   txbuf       ; to transmit buffer
0077 0951           00274          call    TX          ; and transmit it
0078 0923           00275          call    BSTOP       ; generate stop bit
                    00276                              ;
0079 0C0A           00277          movlw   .10
007A 0035           00278          movwf   loops       ; set delay time to give
007B 0901           00279          call    WAIT        ; 10 ms wait after every byte
007C 02AC           00280          incf    addr, F     ; add 1 to low address counter
007D 02F2           00281          decfsz  bcount, F   ; all 8 bytes written?
007E 0A6B           00282          goto    byte        ; no, do another byte
                    00283                              ;
007F 0A62           00284          goto    WRBYTE      ; start over
                    00285 ;
                    00286 ;
                    00287      END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
01C0 : ---------------- ---------------- ---------------- --------------X

All other memory blocks unused.

Program Memory Words Used:   129
Program Memory Words Free:   383
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C: 64KDPOLL.ASM

```
MPASM 01.40 Released          64KDPOLL.ASM   1-16-1997  16:31:49          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54
                     00002 ;*****************************************************************
                     00003 ;       64K Byte Write With Data Polling Program (133 bytes)
                     00004 ;
                     00005 ;       This program demonstrates how to interface a
                     00006 ;       Microchip PIC 16C54 to a 24LC65 Serial E≤ device.
                     00007 ;       This program performs a byte write operation on 8
                     00008 ;       consecutive addresses using the data polling method
                     00009 ;       to determine when the write cycle is complete.
                     00010 ;
                     00011 ;       When writing to a serial E≤ device, there are 2
                     00012 ;       ways to handle the internal timed write cycle
                     00013 ;       time of the device.  The simplest method is
                     00014 ;       simply to wait until the maximum cycle time
                     00015 ;       is exceeded before attempting another command.
                     00016 ;       The other, more efficient method is known as "data
                     00017 ;       polling."  Data polling is done by sending a start
                     00018 ;       bit and control byte to the part after the write
                     00019 ;       cycle has been initiated by a stop bit.  If the ack bit
                     00020 ;       is low, then the device is through writing, otherwise
                     00021 ;       the the sequence is repeated.  If no low acknowledge
                     00022 ;       is found within 40 attempts (about 10 milliseconds)
                     00023 ;       then the routine times out and sets the timeout
                     00024 ;       LED (pin 1) high.
                     00025 ;
                     00026 ;       As an option, the user can connect a LED to pin 1
                     00027 ;       on the PIC (use about a 1K resistor in series) as a
                     00028 ;       timeout indicator.  This LED will come on if the
                     00029 ;       data polling is unsuccessful and the device being
                     00030 ;       programmed does not respond.  This can be tested by
                     00031 ;       simply running the program with no part in the socket.
                     00032 ;
                     00033 ;       Timing is based on using the PIC in 'XT' mode
                     00034 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00035 ;       will be approximately 40 Khz for this setup.
                     00036 ;
                     00037 ;        PIC to Serial EE Connections:
                     00038 ;
                     00039 ;           PIC          Serial EE
                     00040 ;       ------------    ------------
                     00041 ;       Pin 12 (RB6) -->  SCLK
                     00042 ;       Pin 13 (RB7) -->  SDATA
                     00043 ;
                     00044 ;       Pin 1  (RA2) -->  Write cycle timeout fail LED (optional)
                     00045 ;
                     00046 ;
                     00047 ;       Program:          64KDPOLL.ASM
                     00048 ;       Revision Date:
                     00049 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00050 ;
                     00051 ;*****************************************************************
                     00052 ;           Register Definitions
                     00053 ;*****************************************************************
  00000005           00054 port_a  equ     5h      ; port 5 (port_a) used LEDs
```

```
00000006          00055 port_b  equ     6h      ; port 6 (port_b) used for data and
                  00056                         ; clock lines
0000000A          00057 eeprom  equ     0ah     ; bit buffer
0000000B          00058 bycnt   equ     0bh     ; byte counter for read mode
0000000C          00059 addr    equ     0ch     ; address counter
0000000D          00060 datai   equ     0dh     ; data input register
0000000E          00061 datao   equ     0eh     ; data output register
0000000F          00062 slave   equ     0fh     ; device address (1010xxx0)
00000010          00063 txbuf   equ     10h     ; transmit buffer
00000011          00064 count   equ     11h     ; bit counter
00000012          00065 bcount  equ     12h     ; byte counter
00000015          00066 loops   equ     15h     ; delay loop counter
00000016          00067 loops2  equ     16h     ; delay loop counter
00000017          00068 pollcnt equ     17h     ; data polling counter
00000018          00069 addr1   equ     18h     ; word 1 address counter
                  00070 ;*****************************************************
                  00071 ;           Bit Definitions
                  00072 ;*****************************************************
00000007          00073 di      equ     7       ; eeprom input bit
00000006          00074 do      equ     6       ; eeprom output bit
00000007          00075 sdata   equ     7       ; serial EE data line (port_b,pin 13)
00000006          00076 sclk    equ     6       ; serail EE clock line (port_b,pin 12)
00000002          00077 timeout equ     2       ; write cycle timeout fail LED, port_a (pin 1)
00000001          00078 ackf    equ     1       ; acknowledge fail LED, port_a (pin 18)
                  00079 ;*****************************************************
                  00080 ;
01FF              00081         org     01ffh   ; set reset vector
01FF 0A5C         00082         goto    PWRUP
0000              00083         org     000h    ;
0000 0A5C         00084         goto    PWRUP
                  00085 ;
                  00086 ;*****************************************************
                  00087 ;       DELAY ROUTINE
                  00088 ;       This routine takes the value in 'loops'
                  00089 ;       and multiplies it times 1 millisecond to
                  00090 ;       determine delay time.
                  00091 ;*****************************************************
0001              00092 WAIT
                  00093 ;
0001 0C6E         00094 top      movlw   .110           ; timing adjustment variable
0002 0036         00095          movwf   loops2
0003 0000         00096 top2     nop                    ; sit and wait
0004 0000         00097          nop
0005 0000         00098          nop
0006 0000         00099          nop
0007 0000         00100          nop
0008 0000         00101          nop
0009 02F6         00102          decfsz  loops2, F      ; inner loops complete?
000A 0A03         00103          goto    top2           ; no, go again
                  00104                                 ;
000B 02F5         00105          decfsz  loops, F       ; outer loops complete?
000C 0A01         00106          goto    top            ; no, go again
000D 0800         00107          retlw   0              ; yes, return from sub
                  00108 ;
                  00109 ;*****************************************************
                  00110 ;       Start Bit Subroutine
                  00111 ;       this routine generates a start bit
                  00112 ;       (Low going data line while clock is high)
                  00113 ;*****************************************************
                  00114 ;
000E              00115 BSTART
000E 05E6         00116          bsf     port_b,sdata   ; make sure data is high
000F 0C3F         00117          movlw   b'00111111'
0010 0006         00118          tris    port_b         ; set data and clock lines for output
0011 04C6         00119          bcf     port_b,sclk    ; make sure clock is low
0012 0000         00120          nop
```


© 1997 Microchip Technology Inc.

```
0013 05C6          00121          bsf     port_b,sclk     ; set clock high
0014 0000          00122          nop
0015 0000          00123          nop
0016 0000          00124          nop
0017 0000          00125          nop
0018 0000          00126          nop
0019 04E6          00127          bcf     port_b,sdata    ; data line goes low during
                   00128                                  ; high clock for start bit
001A 0000          00129          nop
001B 0000          00130          nop
001C 0000          00131          nop
001D 0000          00132          nop
001E 0000          00133          nop                    ; timing adjustment
001F 04C6          00134          bcf     port_b,sclk    ; start clock train
0020 0000          00135          nop
0021 0000          00136          nop
0022 0800          00137          retlw   0
                   00138          ;
                   00139 ;*************************************************************
                   00140 ;       Stop Bit Subroutine
                   00141 ;       This routine generates a stop bit
                   00142 ;       (High going data line while clock is high)
                   00143 ;*************************************************************
0023               00144 BSTOP
0023 0C3F          00145          movlw   b'00111111'    ;
0024 0006          00146          tris    port_b         ; set data/clock lines as outputs
0025 04E6          00147          bcf     port_b,sdata   ; make sure data line is low
0026 0000          00148          nop
0027 0000          00149          nop
0028 0000          00150          nop
0029 05C6          00151          bsf     port_b,sclk    ; set clock high
002A 0000          00152          nop
002B 0000          00153          nop
002C 0000          00154          nop
002D 05E6          00155          bsf     port_b,sdata   ; data goes high while clock high
                   00156                                 ; for stop bit
002E 0000          00157          nop
002F 0000          00158          nop
0030 04C6          00159          bcf     port_b,sclk    ; set clock low again
0031 0000          00160          nop
0032 0000          00161          nop
0033 0000          00162          nop
0034 0800          00163          retlw   0
                   00164 ;
                   00165 ;*************************************************************
                   00166 ;       BITOUT routine takes one bit of data in 'do' and
                   00167 ;       transmits it to the serial EE device
                   00168 ;*************************************************************
0035               00169 BITOUT
0035 0C3F          00170          movlw   b'00111111'    ; set data,clock as outputs
0036 0006          00171          tris    port_b
0037 07CA          00172          btfss   eeprom,do      ; check for state of data bit to xmit
0038 0A3B          00173          goto    bitlow         ;
0039 05E6          00174          bsf     port_b,sdata   ; set data line high
003A 0A3C          00175          goto    clkout         ; go toggle the clock
                   00176                                 ;
003B 04E6          00177 bitlow   bcf     port_b,sdata   ; output a low bit
003C 05C6          00178 clkout   bsf     port_b,sclk    ; set clock line high
003D 0000          00179          nop
003E 0000          00180          nop
003F 0000          00181          nop
0040 0000          00182          nop
0041 04C6          00183          bcf     port_b,sclk    ; return clock line low
0042 0800          00184          retlw   0
                   00185 ;
                   00186 ;                   End of Subroutine
```

7

© 1997 Microchip Technology Inc.

```
                    00187 ;
                    00188 ;*****************************************************************
                    00189 ;       BITIN routine reads one bit of data from the
                    00190 ;       serial EE device and stores it in 'di'
                    00191 ;*****************************************************************
0043                00192 BITIN
0043 05EA           00193          bsf     eeprom,di       ; assume input bit is high
0044 0CBF           00194          movlw   b'10111111'     ; make sdata an input line
0045 0006           00195          tris    port_b
0046 05E6           00196          bsf     port_b,sdata    ; set sdata line for input
0047 05C6           00197          bsf     port_b,sclk     ; set clock line high
0048 0000           00198          nop                     ; just sit here a sec
0049 0000           00199          nop
004A 0000           00200          nop
004B 0000           00201          nop
004C 0000           00202          nop                     ;
004D 07E6           00203          btfss   port_b,sdata    ; read the data bit
004E 04EA           00204          bcf     eeprom,di       ; input bit was low
004F 04C6           00205          bcf     port_b,sclk     ; set clock line low
                    00206                                  ;
0050 0800           00207          retlw   0               ;
                    00208 ;
                    00209 ;*****************************************************************
                    00210 ;       Transmit Data Subroutine
                    00211 ;       This routine takes the byte of data stored in the
                    00212 ;       'datao' register and transmits it to the serial EE device.
                    00213 ;       It will then send 1 more clock to the serial EE for the
                    00214 ;       acknowledge bit.  If the ack bit from the part was low
                    00215 ;       then the transmission was sucessful.  If it is high, then
                    00216 ;       the device did not send a proper ack bit and the ack
                    00217 ;       fail LED will be turned on.
                    00218 ;*****************************************************************
0051                00219 TX
0051 0C08           00220          movlw   .8
0052 0031           00221          movwf   count           ; set the #bits to 8
                    00222          ;
0053                00223 TXLP
0053 04CA           00224          bcf     eeprom,do       ; assume bit out is low
0054 06F0           00225          btfsc   txbuf,7         ; is bit out really low?
0055 05CA           00226          bsf     eeprom,do       ; otherwise data bit =1
0056 0935           00227          call    BITOUT          ; serial data out
0057 0370           00228          rlf     txbuf, F        ; rotate txbuf left
0058 02F1           00229          decfsz  count, F        ; 8 bits done?
0059 0A53           00230          goto    TXLP            ; no - go again
005A 0943           00231          call    BITIN           ; read ack bit
                    00232          ;
005B 0800           00233          retlw   0
                    00234 ;                    ;
                    00235 ;*****************************************************************
                    00236 ;       Power up routine
                    00237 ;       This is the program entry point, which in this case simply
                    00238 ;       sets the port_a I/O lines and directs control to the
                    00239 ;       write routine.
                    00240 ;*****************************************************************
005C                00241 PWRUP
005C 0C00           00242           movlw  b'00000000'
005D 0005           00243           tris   port_a           ; set port A as all output
005E 0065           00244           clrf   port_a           ; all output lines low
005F 0A60           00245           goto   WRBYTE
                    00246
                    00247 ;*****************************************************************
                    00248 ;       Byte Write Routine with data polling technique
                    00249 ;
                    00250 ;       This routine writes the data in "datao" to
                    00251 ;       8 consecutive bytes in the serial EE device starting
                    00252 ;       at address 00. To determine when the write cycle time
```


© 1997 Microchip Technology Inc.

```
                    00253 ;       of the device, the 'data polling' method is used.  This
                    00254 ;       involves sending a start bit and control byte to the part
                    00255 ;       and checking for an acknowledge.  If the ack bit is low, then
                    00256 ;       the device is through writing, otherwise the the sequence
                    00257 ;       is repeated.  If no low acknowledge is found within 40
                    00258 ;       attempts (about 10 milliseconds) then the routine times
                    00259 ;       out and sets the timeout LED (pin 18) high.  This program
                    00260 ;       will repeat forever.
                    00261 ;
                    00262 ;*****************************************************************
                    00263 ;
0060                00264 WRBYTE
                    00265 ;
0060 0065           00266         clrf    port_a          ; all LEDs off
0061 0CA0           00267         movlw   b'10100000'     ; set slave address and write mode
0062 002F           00268         movwf   slave
0063 0CAA           00269         movlw   b'10101010'     ; set data to write as AAh
0064 002E           00270         movwf   datao
0065 0C08           00271         movlw   .8              ; set number of bytes
0066 0032           00272         movwf   bcount          ; to write as to 8
0067 0078           00273         clrf    addr1           ; set starting high address to 00
0068 006C           00274         clrf    addr            ; set starting low address to 00
                    00275                                 ;
0069 090E           00276 byte    call    BSTART          ; generate start bit
006A 020F           00277         movf    slave,w         ; move slave address
006B 0030           00278         movwf   txbuf           ; into transmit buffer
006C 0951           00279         call    TX              ; and send it
006D 0218           00280         movf    addr1,w         ; move word 1 address
006E 0030           00281         movwf   txbuf           ; into transmit buffer
006F 0951           00282         call    TX              ; and send it
0070 020C           00283         movf    addr,w          ; move word 0 address
0071 0030           00284         movwf   txbuf           ; into transmit buffer
0072 0951           00285         call    TX              ; and send it
0073 020E           00286         movf    datao,w         ; move data byte
0074 0030           00287         movwf   txbuf           ; to tranmit buffer
0075 0951           00288         call    TX              ; and transmit it
0076 0923           00289         call    BSTOP           ; generate stop bit
                    00290                                 ;
                    00291                                 ; now start polling for a low ack bit
                    00292                                 ;
0077 0C28           00293         movlw   .40             ;
0078 0037           00294         movwf   pollcnt         ; set max number of times to poll as 40
0079 090E           00295 poll    call    BSTART          ; generate start bit
007A 0CA0           00296         movlw   b'10100000'     ; move slave address (write mode)
007B 0030           00297         movwf   txbuf           ; into transmit buffer
007C 0951           00298         call    TX              ; and send it
007D 07EA           00299         btfss   eeprom,di       ; was the ack bit low?
007E 0A82           00300         goto    exitpoll        ; yes, do another byte
007F 02F7           00301         decfsz  pollcnt, F      ; is poll counter down to zero?
0080 0A79           00302         goto    poll            ; no, poll again. Otherwise the part is
0081 0545           00303         bsf     port_a,timeout  ; not responding in time so set timeout
                    00304                                 ; LED and continue on
                    00305                                 ;
0082 02AC           00306 exitpoll incf   addr, F         ; add 1 to address counter
0083 02F2           00307         decfsz  bcount, F       ; all 8 bytes written?
0084 0A69           00308         goto    byte            ; no, do another byte
0085 0A60           00309         goto    WRBYTE          ; yes, start over
                    00310 ;
                    00311 ;
                    00312     END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXX---------- ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X
```

7

```
All other memory blocks unused.

Program Memory Words Used:   135
Program Memory Words Free:   377
Errors    :     0
Warnings  :     0 reported,     0 suppressed
Messages  :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX D: 64KPAGEW.ASM

```
MPASM 01.40 Released          64KPAGEW.ASM   1-16-1997  16:32:22          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54
                     00002 ;*****************************************************************
                     00003 ;       64K Page Write Program (126 bytes)
                     00004 ;
                     00005 ;       The 24LC65 has a page length of 8 bytes.  This page
                     00006 ;       can be used to write up to 8 bytes of data into the
                     00007 ;       part before initiating the write cycle.  Since the
                     00008 ;       write cycle is timed the same for 1 byte or 8 bytes
                     00009 ;       it is more efficient to use the page mode when
                     00010 ;       consecutive addresses are being written to.
                     00011 ;
                     00012 ;       When using page mode, the control byte, upper and lower
                     00013 ;       addresses are sent for the first address only. After the
                     00014 ;       data byte for the first address is sent, the data for the
                     00015 ;       next consecutive address is clocked in.  This is
                     00016 ;       repeated as many times as needed (as long as the page
                     00017 ;       length is not exceeded) and then a stop bit is sent.
                     00018 ;       The device will still acknowledge between every byte of
                     00019 ;       data.  After the stop bit is sent, the part will
                     00020 ;       initiate the self timed write cycle.
                     00021 ;
                     00022 ;       This routine waits approximately 10mS for the write
                     00023 ;       cycle to complete for each page.  A more efficient
                     00024 ;       method of determining when the write cycle is complete
                     00025 ;       is called "data polling."  The data polling method is
                     00026 ;       explained in the program "64kdpoll.asm."  That
                     00027 ;       particular program uses data polling in a byte write
                     00028 ;       mode but it can be used in exactly the same way for
                     00029 ;       a page mode write.
                     00030 ;
                     00031 ;       As an option, the user can connect a LED to pin 18
                     00032 ;       on the PIC (use about a 1K resistor in series) as a
                     00033 ;       acknowledge fail indicator.  This LED will come on
                     00034 ;       if the device being programmed does not send a low
                     00035 ;       acknowledge bit at the proper times.  This can be
                     00036 ;       tested by simply running the program with no part
                     00037 ;       in the socket.
                     00038 ;
                     00039 ;       Timing is based on using the PIC in 'XT' mode
                     00040 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00041 ;       will be approximately 40 Khz for this setup.
                     00042 ;
                     00043 ;
                     00044 ;        PIC to Serial EE Connections:
                     00045 ;
                     00046 ;            PIC          Serial EE
                     00047 ;       ------------    ------------
                     00048 ;       Pin 12 (RB6) -->  SCLK
                     00049 ;       Pin 13 (RB7) -->  SDATA
                     00050 ;
                     00051 ;       PIN 18 (RA1) -->  Acknowledge fail LED  (Optional)
                     00052 ;
                     00053 ;
                     00054 ;
```


© 1997 Microchip Technology Inc.

```
                    00055 ;       Program:          64KPAGEW.ASM
                    00056 ;       Revision Date:
                    00057 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                    00058 ;
                    00059 ;***********************************************************
                    00060 ;           Register Definitions
                    00061 ;***********************************************************
  00000005          00062 port_a  equ     5h      ; port 5 (port_a) used for LEDs
  00000006          00063 port_b  equ     6h      ; port 6 (port b) used for data and
                    00064                         ; clock lines
  0000000A          00065 eeprom  equ     0ah     ; bit buffer
  0000000B          00066 bycnt   equ     0bh     ; byte counter for read mode
  0000000C          00067 addr    equ     0ch     ; word 0 address counter
  0000000D          00068 datai   equ     0dh     ; data input register
  0000000E          00069 datao   equ     0eh     ; data output register
  0000000F          00070 slave   equ     0fh     ; device address (1010xxx0)
  00000010          00071 txbuf   equ     10h     ; transmit buffer
  00000011          00072 count   equ     11h     ; bit counter
  00000012          00073 bcount  equ     12h     ; byte counter
  00000015          00074 loops   equ     15h     ; delay loop counter
  00000016          00075 loops2  equ     16h     ; delay loop counter
  00000017          00076 addr1   equ     17h     ; word 1 address counter
                    00077 ;***********************************************************
                    00078 ;           Bit Definitions
                    00079 ;***********************************************************
  00000007          00080 di      equ     7       ; eeprom input bit
  00000006          00081 do      equ     6       ; eeprom output bit
  00000007          00082 sdata   equ     7       ; serial EE data line (port_b,pin 13)
  00000006          00083 sclk    equ     6       ; serail EE clock line (port_b,pin 12)
  00000001          00084 ackf    equ     1       ; acknowledge fail LED (port_a,pin 18)
                    00085 ;***********************************************************
                    00086 ;
                    00087 ;
01FF                00088         org     01ffh   ; set reset vector
01FF 0A5E           00089         goto    PWRUP
0000                00090         org     000h    ;
0000 0A5E           00091         goto    PWRUP
                    00092 ;
                    00093 ;***********************************************************
                    00094 ;       DELAY ROUTINE
                    00095 ;       This routine takes the value in 'loops'
                    00096 ;       and multiplies it times 1 millisecond to
                    00097 ;       determine delay time.
                    00098 ;***********************************************************
0001                00099 WAIT
                    00100 ;
0001 0C6E           00101 top       movlw   .110            ; timing adjustment variable
0002 0036           00102           movwf   loops2
0003 0000           00103 top2      nop                     ; sit and wait
0004 0000           00104           nop
0005 0000           00105           nop
0006 0000           00106           nop
0007 0000           00107           nop
0008 0000           00108           nop
0009 02F6           00109           decfsz  loops2, F       ; inner loops complete?
000A 0A03           00110           goto    top2            ; no, go again
                    00111                                   ;
000B 02F5           00112           decfsz  loops, F        ; outer loops complete?
000C 0A01           00113           goto    top             ; no, go again
000D 0800           00114           retlw   0               ; yes, return from sub
                    00115 ;
                    00116 ;***********************************************************
                    00117 ;       Start Bit Subroutine
                    00118 ;       this routine generates a start bit
                    00119 ;       (Low going data line while clock is high)
                    00120 ;***********************************************************
```


© 1997 Microchip Technology Inc.

```
                    00121 ;
000E                00122 BSTART
000E 05E6           00123           bsf     port_b,sdata    ; make sure data is high
000F 0C3F           00124           movlw   b'00111111'
0010 0006           00125           tris    port_b          ; set data and clock lines for output
0011 04C6           00126           bcf     port_b,sclk     ; make sure clock is low
0012 0000           00127           nop
0013 05C6           00128           bsf     port_b,sclk     ; set clock high
0014 0000           00129           nop
0015 0000           00130           nop
0016 0000           00131           nop
0017 0000           00132           nop
0018 0000           00133           nop
0019 04E6           00134           bcf     port_b,sdata    ; data line goes low during
                    00135                                   ; high clock for start bit
001A 0000           00136           nop
001B 0000           00137           nop
001C 0000           00138           nop
001D 0000           00139           nop
001E 0000           00140           nop                     ; timing adjustment
001F 04C6           00141           bcf     port_b,sclk     ; start clock train
0020 0000           00142           nop
0021 0000           00143           nop
0022 0800           00144           retlw   0
                    00145 ;
                    00146 ;*************************************************************
                    00147 ;       Stop Bit Subroutine
                    00148 ;       This routine generates a stop bit
                    00149 ;       (High going data line while clock is high)
                    00150 ;*************************************************************
0023                00151 BSTOP
0023 0C3F           00152           movlw   b'00111111'     ;
0024 0006           00153           tris    port_b          ; set data/clock lines as outputs
0025 04E6           00154           bcf     port_b,sdata    ; make sure data line is low
0026 0000           00155           nop
0027 0000           00156           nop
0028 0000           00157           nop
0029 05C6           00158           bsf     port_b,sclk     ; set clock high
002A 0000           00159           nop
002B 0000           00160           nop
002C 0000           00161           nop
002D 05E6           00162           bsf     port_b,sdata    ; data goes high while clock high
                    00163                                   ; for stop bit
002E 0000           00164           nop
002F 0000           00165           nop
0030 04C6           00166           bcf     port_b,sclk     ; set clock low again
0031 0000           00167           nop
0032 0000           00168           nop
0033 0000           00169           nop
0034 0800           00170           retlw   0
                    00171 ;
                    00172 ;*************************************************************
                    00173 ;       BITOUT routine takes one bit of data in 'do' and
                    00174 ;       transmits it to the serial EE device
                    00175 ;*************************************************************
0035                00176 BITOUT
0035 0C3F           00177           movlw   b'00111111'     ; set data,clock as outputs
0036 0006           00178           tris    port_b
0037 07CA           00179           btfss   eeprom,do       ; check for state of data bit to xmit
0038 0A3B           00180           goto    bitlow          ;
0039 05E6           00181           bsf     port_b,sdata    ; set data line high
003A 0A3C           00182           goto    clkout          ; go toggle the clock
                    00183
003B 04E6           00184 bitlow    bcf     port_b,sdata    ; output a low bit
003C 05C6           00185 clkout    bsf     port_b,sclk     ; set clock line high
003D 0000           00186           nop
```

7

```
003E 0000           00187          nop
003F 0000           00188          nop
0040 0000           00189          nop
0041 04C6           00190          bcf     port_b,sclk     ; return clock line low
0042 0800           00191          retlw   0
                    00192 ;
                    00193 ;*****************************************************************
                    00194 ;       BITIN routine reads one bit of data from the
                    00195 ;       serial EE device and stores it in 'di'
                    00196 ;*****************************************************************
0043                00197 BITIN
0043 05EA           00198          bsf     eeprom,di       ; assume input bit is high
0044 0CBF           00199          movlw   b'10111111'     ; make sdata an input line
0045 0006           00200          tris    port_b
0046 05E6           00201          bsf     port_b,sdata    ; set sdata line for input
0047 05C6           00202          bsf     port_b,sclk     ; set clock line high
0048 0000           00203          nop                     ; just sit here a sec
0049 0000           00204          nop
004A 0000           00205          nop
004B 0000           00206          nop
004C 0000           00207          nop                     ;
004D 07E6           00208          btfss   port_b,sdata    ; read the data bit
004E 04EA           00209          bcf     eeprom,di       ; input bit was low
004F 04C6           00210          bcf     port_b,sclk     ; set clock line low
                    00211                                  ;
0050 0800           00212          retlw   0               ;
                    00213 ;
                    00214 ;*****************************************************************
                    00215 ;       Transmit Data Subroutine
                    00216 ;       This routine takes the byte of data stored in the
                    00217 ;       'datao' register and transmits it to the serial EE device.
                    00218 ;       It will then send 1 more clock to the serial EE for the
                    00219 ;       acknowledge bit.  If the ack bit from the part was low
                    00220 ;       then the transmission was sucessful.  If it is high, then
                    00221 ;       the device did not send a proper ack bit and the ack
                    00222 ;       fail LED will be turned on.
                    00223 ;*****************************************************************
0051                00224 TX
0051 0C08           00225          movlw   .8
0052 0031           00226          movwf   count           ; set the #bits to 8
                    00227          ;
0053                00228 TXLP
0053 04CA           00229          bcf     eeprom,do       ; assume bit out is low
0054 06F0           00230          btfsc   txbuf,7         ; is bit out really low?
0055 05CA           00231          bsf     eeprom,do       ; otherwise data bit =1
0056 0935           00232          call    BITOUT          ; serial data out
0057 0370           00233          rlf     txbuf, F        ; rotate txbuf left
0058 02F1           00234          decfsz  count, F        ; 8 bits done?
0059 0A53           00235          goto    TXLP            ; no - go again
005A 0943           00236          call    BITIN           ; read ack bit
005B 06EA           00237          btfsc   eeprom,di       ; check ack bit
005C 0525           00238          bsf     port_a,ackf     ; set acknowledge fail LED if the
                    00239          ;
005D 0800           00240          retlw   0
                    00241 ;
                    00242 ;*****************************************************************
                    00243 ;       Power up routine
                    00244 ;       This is the program entry point, which in this case simply
                    00245 ;       sets the port_a I/O lines and directs control to the
                    00246 ;       byte write routine.
                    00247 ;*****************************************************************
005E                00248 PWRUP
005E 0C00           00249           movlw  b'00000000'
005F 0005           00250          tris    port_a          ; set port A as all output
0060 0065           00251          clrf    port_a          ; all output lines low
0061 0A62           00252          goto    pwrite          ; go do the page write
```


© 1997 Microchip Technology Inc.

```
                    00253
                    00254 ;*****************************************************************
                    00255 ;       Page Write Routine
                    00256 ;       This routine writes the data in "datao" to 8 consecutive
                    00257 ;       bytes using page write mode starting at address 00.
                    00258 ;       This routine waits 10mS after every page to give the device
                    00259 ;       time to do the write.  This program repeats forever.
                    00260 ;*****************************************************************
                    00261 ;
0062                00262 pwrite
                    00263 ;
0062 0065           00264           clrf    port_a      ; clear all LEDs
0063 0CA0           00265           movlw   b'10100000' ; set slave address and write mode
0064 002F           00266           movwf   slave
0065 0C55           00267           movlw   b'01010101' ; set data to write as 55h
0066 002E           00268           movwf   datao
                    00269 ;
0067 0C08           00270           movlw   .8          ; set number of bytes
0068 0032           00271           movwf   bcount      ; to write as to 8
0069 0077           00272           clrf    addr1       ; set high address byte to 00
006A 006C           00273           clrf    addr        ; set low address byte to 00
                    00274                               ;
006B 090E           00275           call    BSTART      ; generate start bit
006C 020F           00276           movf    slave,w     ; move slave address
006D 0030           00277           movwf   txbuf       ; into transmit buffer
006E 0951           00278           call    TX          ; and send it
006F 0217           00279           movf    addr1,w     ; move word 1 address
0070 0030           00280           movwf   txbuf       ; into transmit buffer
0071 0951           00281           call    TX          ; and send it
0072 020C           00282           movf    addr,w      ; move word 0 address
0073 0030           00283           movwf   txbuf       ; into transmit buffer
0074 0951           00284           call    TX          ; and send it
                    00285
0075 020E           00286 byte      movf    datao,w     ; move data byte
0076 0030           00287           movwf   txbuf       ; to transmit buffer
0077 0951           00288           call    TX          ; and transmit it
0078 02F2           00289           decfsz  bcount, F   ; all 8 bytes written?
0079 0A75           00290           goto    byte        ; no, do another byte
                    00291                               ;
007A 0923           00292           call    BSTOP       ; generate stop bit
007B 0C0A           00293           movlw   .10
007C 0035           00294           movwf   loops       ; set delay time to give
007D 0901           00295           call    WAIT        ; 10 ms wait after every byte
007E 0A62           00296           goto    pwrite      ; start over
                    00297 ;
                    00298 ;
                    00299     END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXX-
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   128
Program Memory Words Free:   384
Errors   :     0
Warnings :     0 reported,      0 suppressed
Messages :     0 reported,      0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX E: 64KSEQR.ASM

```
MPASM 01.40 Released          64KSEQR.ASM   1-16-1997  16:32:46         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54
                     00002 ;*****************************************************************
                     00003 ;       64K Sequential Read Program (142 bytes)
                     00004 ;
                     00005 ;       This program demonstrates how to interface a
                     00006 ;       Microchip PIC 16C54 to a 24LC65 Serial E≤ device
                     00007 ;       and perform a sequential read operation. This program
                     00008 ;       will read 8 consecutive addresses in the 'sequential
                     00009 ;       read' mode.  This entails setting the address pointer
                     00010 ;       for the first address only and then clocking out as many
                     00011 ;       bytes of data as needed.  The device will automatically
                     00012 ;       increment the address.
                     00013 ;
                     00014 ;       Timing is based on using the PIC in 'XT' mode
                     00015 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00016 ;       will be approximately 60 Khz for this setup.
                     00017 ;
                     00018 ;       As an option, the user may connect a LED to pin 18
                     00019 ;       on the PIC (use about a 1K resistor in series) as an
                     00020 ;       acknowledge fail indicator.  This LED will come on if
                     00021 ;       the serial EE fails to acknowledge correctly.
                     00022 ;
                     00023 ;        PIC to Serial EE Connections:
                     00024 ;
                     00025 ;           PIC          Serial EE
                     00026 ;       ------------    ------------
                     00027 ;       Pin 12 (RB6) -->  SCLK
                     00028 ;       Pin 13 (RB7) -->  SDATA
                     00029 ;
                     00030 ;       PIN 18 (RA1) -->  Acknowledge fail LED  (Optional)
                     00031 ;
                     00032 ;
                     00033 ;
                     00034 ;       Program:          64KSEQR.ASM
                     00035 ;       Revision Date:
                     00036 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00037 ;
                     00038 ;*****************************************************************
                     00039 ;           Register Definitions
                     00040 ;*****************************************************************
00000005             00041 port_a  equ     5h      ; port 5 (port a) used for LED display
00000006             00042 port_b  equ     6h      ; port 6 (port b) used for data and
                     00043                         ; clock lines
0000000A             00044 eeprom  equ     0ah     ; bit buffer
0000000B             00045 bycnt   equ     0bh     ; byte counter for read mode
0000000C             00046 addr    equ     0ch     ; word 0 address counter
0000000D             00047 datai   equ     0dh     ; data input register
0000000E             00048 datao   equ     0eh     ; data output register
0000000F             00049 slave   equ     0fh     ; device address 1010xxx0)
00000010             00050 txbuf   equ     10h     ; transmit buffer
00000011             00051 count   equ     11h     ; bit counter
00000012             00052 bcount  equ     12h     ; byte counter
00000015             00053 loops   equ     15h     ; delay loop counter
00000016             00054 loops2  equ     16h     ; delay loop counter
```


© 1997 Microchip Technology Inc.

```
00000017          00055 addr1   equ    17h     ; word 1 address counter
                  00056 ;*****************************************************
                  00057 ;           Bit Definitions
                  00058 ;*****************************************************
00000007          00059 di      equ    7       ; eeprom input bit
00000006          00060 do      equ    6       ; eeprom output bit
00000007          00061 sdata   equ    7       ; serial EE data line (port_b,pin 13)
00000006          00062 sclk    equ    6       ; serail EE clock line (port_b,pin 12)
00000001          00063 ackf    equ    1       ; acknowledge fail LED (port_a)
                  00064 ;*****************************************************
                  00065 ;
01FF              00066         org    01ffh   ; set reset vector
01FF 0A69         00067         goto   PWRUP
0000              00068         org    000h    ;
0000 0A69         00069         goto   PWRUP
                  00070 ;
                  00071 ;*****************************************************
                  00072 ;       DELAY ROUTINE
                  00073 ;       This routine takes the value in 'loops'
                  00074 ;       and multiplies it times 1 millisecond to
                  00075 ;       determine delay time.
                  00076 ;*****************************************************
0001              00077 WAIT
                  00078 ;
0001 0C6E         00079 top      movlw   .110           ; timing adjustment variable
0002 0036         00080          movwf   loops2
0003 0000         00081 top2     nop                    ; sit and wait
0004 0000         00082          nop
0005 0000         00083          nop
0006 0000         00084          nop
0007 0000         00085          nop
0008 0000         00086          nop
0009 02F6         00087          decfsz  loops2, F      ; inner loops complete?
000A 0A03         00088          goto    top2           ; no, go again
                  00089                                 ;
000B 02F5         00090          decfsz  loops, F       ; outer loops complete?
000C 0A01         00091          goto    top            ; no, go again
000D 0800         00092          retlw   0              ; yes, return from sub
                  00093 ;
                  00094 ;*****************************************************
                  00095 ;       Start Bit Subroutine
                  00096 ;       this routine generates a start bit
                  00097 ;       (Low going data line while clock is high)
                  00098 ;*****************************************************
                  00099 ;
000E              00100 BSTART
000E 05E6         00101          bsf     port_b,sdata   ; make sure data is high
000F 0C3F         00102          movlw   b'00111111'
0010 0006         00103          tris    port_b         ; set data and clock lines for output
0011 04C6         00104          bcf     port_b,sclk    ; make sure clock is low
0012 0000         00105          nop
0013 05C6         00106          bsf     port_b,sclk    ; set clock high
0014 0000         00107          nop
0015 0000         00108          nop
0016 0000         00109          nop
0017 0000         00110          nop
0018 0000         00111          nop
0019 04E6         00112          bcf     port_b,sdata   ; data line goes low during
                  00113                                 ; high clock for start bit
001A 0000         00114          nop
001B 0000         00115          nop
001C 0000         00116          nop
001D 0000         00117          nop
001E 0000         00118          nop                    ; timing adjustment
001F 04C6         00119          bcf     port_b,sclk    ; start clock train
0020 0000         00120          nop
```

7

```
0021 0000           00121          nop
0022 0800           00122          retlw   0
                    00123          ;
                    00124 ;*******************************************************
                    00125 ;       Stop Bit Subroutine
                    00126 ;       This routine generates a stop bit
                    00127 ;       (High going data line while clock is high)
                    00128 ;*******************************************************
0023                00129 BSTOP
0023 04E6           00130          bcf     port_b,sdata    ; make sure data line is low
0024 0C3F           00131          movlw   b'00111111'     ;
0025 0006           00132          tris    port_b          ; set data/clock lines as outputs
0026 04E6           00133          bcf     port_b,sdata    ; make sure data line is low
0027 0000           00134          nop
0028 0000           00135          nop
0029 0000           00136          nop
002A 05C6           00137          bsf     port_b,sclk     ; set clock high
002B 0000           00138          nop
002C 0000           00139          nop
002D 0000           00140          nop
002E 05E6           00141          bsf     port_b,sdata    ; data goes high while clock high
                    00142                                  ; for stop bit
002F 0000           00143          nop
0030 0000           00144          nop
0031 04C6           00145          bcf     port_b,sclk     ; set clock low again
0032 0000           00146          nop
0033 0000           00147          nop
0034 0000           00148          nop
0035 0800           00149          retlw   0
                    00150 ;
                    00151 ;*******************************************************
                    00152 ;       BITOUT routine takes the bit of data in 'do' and
                    00153 ;       transmits it to the serial EE device
                    00154 ;*******************************************************
0036                00155 BITOUT
0036 0C3F           00156          movlw   b'00111111'     ; set data,clock as outputs
0037 0006           00157          tris    port_b
0038 07CA           00158          btfss   eeprom,do       ; check for state of data bit to xmit
0039 0A3C           00159          goto    bitlow          ; low? go set data line low
003A 05E6           00160          bsf     port_b,sdata    ; high? set data line high
003B 0A3D           00161          goto    clkout          ; go toggle the clock
                    00162
003C 04E6           00163 bitlow   bcf     port_b,sdata    ; output a low bit
003D 05C6           00164 clkout   bsf     port_b,sclk     ; set clock line high
003E 0000           00165          nop
003F 0000           00166          nop
0040 0000           00167          nop
0041 0000           00168          nop
0042 04C6           00169          bcf     port_b,sclk     ; return clock line low
0043 0800           00170          retlw   0
                    00171 ;
                    00172 ;*******************************************************
                    00173 ;       BITIN routine reads one bit of data from the
                    00174 ;       serial EE device and stores it in the bit 'di'
                    00175 ;*******************************************************
0044                00176 BITIN
0044 05EA           00177          bsf     eeprom,di       ; assume input bit is high
0045 0CBF           00178          movlw   b'10111111'     ; make sdata an input line
0046 0006           00179          tris    port_b
0047 05E6           00180          bsf     port_b,sdata    ; set sdata line for input
0048 05C6           00181          bsf     port_b,sclk     ; set clock line high
0049 0000           00182          nop                     ; just sit here a sec
004A 0000           00183          nop
004B 0000           00184          nop
004C 0000           00185          nop
004D 0000           00186          nop
```


© 1997 Microchip Technology Inc.

```
004E 07E6          00187          btfss   port_b,sdata    ; read the data bit
004F 04EA          00188          bcf     eeprom,di       ; input bit was low, set 'di' accordingly
0050 04C6          00189          bcf     port_b,sclk     ; set clock line low
                   00190                                  ;
0051 0800          00191          retlw   0               ;
                   00192 ;
                   00193 ;*****************************************************************
                   00194 ;       Transmit Data Subroutine
                   00195 ;       This routine takes the byte of data stored in the
                   00196 ;       'datao' register and transmits it to the serial EE device.
                   00197 ;       It will then send 1 more clock to the serial EE for the
                   00198 ;       acknowledge bit.  If the ack bit from the part was low
                   00199 ;       then the transmission was sucessful.  If it is high, then
                   00200 ;       the device did not send a proper ack bit and the ack
                   00201 ;       fail LED will be turned on.
                   00202 ;*****************************************************************
0052               00203 TX
0052 0C08          00204          movlw   .8
0053 0031          00205          movwf   count           ; set the #bits to 8
                   00206          ;
0054               00207 TXLP
0054 04CA          00208          bcf     eeprom,do       ; assume bit out is low
0055 06F0          00209          btfsc   txbuf,7         ; is bit out really low?
0056 05CA          00210          bsf     eeprom,do       ; no, set it high
0057 0936          00211          call    BITOUT          ; send the bit to serial EE
0058 0370          00212          rlf     txbuf, F        ; rotate txbuf left
0059 02F1          00213          decfsz  count, F        ; 8 bits done?
005A 0A54          00214          goto    TXLP            ; no - go again
005B 0944          00215          call    BITIN           ; read ack bit
005C 06EA          00216          btfsc   eeprom,di       ; check ack bit
005D 0525          00217          bsf     port_a,ackf     ; set acknowledge fail LED if the
                   00218                                  ; device did not pull data low
                   00219          ;
005E 0800          00220          retlw   0
                   00221 ;
                   00222 ;*****************************************************************
                   00223 ;       Receive data routine
                   00224 ;       This routine reads one byte of data from the part
                   00225 ;       into the 'datai' register.  It then sends a high
                   00226 ;       ack bit to indicate that no more data is to be read
                   00227 ;*****************************************************************
005F               00228 RX
005F 006D          00229          clrf    datai           ; clear input buffer
0060 0C08          00230          movlw   .8              ; set # bits to 8
0061 0031          00231          movwf   count
0062 036D          00232 RXLP     rlf     datai, F        ; rotate datai 1 bit left
0063 0944          00233          call    BITIN           ; read a bit
0064 06EA          00234          btfsc   eeprom,di
0065 050D          00235          bsf     datai,0         ; set bit 0 if necessary
0066 02F1          00236          decfsz  count, F        ; 8 bits done?
0067 0A62          00237          goto    RXLP            ; no, do another
0068 0800          00238          retlw   0
                   00239 ;
                   00240 ;*****************************************************************
                   00241 ;       Power up routine
                   00242 ;       This is the program entry point.  I/O line status is
                   00243 ;       set for port A here.
                   00244 ;*****************************************************************
0069               00245 PWRUP
0069 0C00          00246           movlw  b'00000000'
006A 0005          00247           tris   port_a           ; set port A as all output
006B 0065          00248           clrf   port_a           ; all output lines low
006C 0A6D          00249           goto   READ
                   00250 ;
                   00251 ;*****************************************************************
                   00252 ;       READ (read routine)
```

7

```
                    00253 ;       This routine reads 8 consecutive addresses of the
                    00254 ;       serial EE device starting at address 00 in the
                    00255 ;       sequential read mode. Reading in this mode is more
                    00256 ;       efficient than the random read mode as the control byte
                    00257 ;       and address have to be sent only once at the beginning
                    00258 ;       of the sequence.  As many consecutive addresses as
                    00259 ;       needed can then be read from the part until a stop bit is
                    00260 ;       sent.  In the read mode, the PIC must send the acknowledge
                    00261 ;       bit after every 8 data bits from the device.  When the
                    00262 ;       last byte needed has been read, then the controller will
                    00263 ;       send a high acknowledge bit and then a stop bit to halt
                    00264 ;       transmission from the device.
                    00265 ;*****************************************************************
006D                00266 READ
                    00267 ;
006D 0425           00268           bcf     port_a,ackf ; clear the ack fail LED if on
006E 0C08           00269           movlw   .8
006F 0032           00270           movwf   bcount      ; set number of bytes to read as 8
0070 0CA0           00271           movlw   b'10100000' ; set slave address and write mode
0071 002F           00272           movwf   slave
0072 0077           00273           clrf    addr1       ; set starting high address to 00
0073 006C           00274           clrf    addr        ; set starting low address to 00
                    00275                               ;
0074 090E           00276           call    BSTART      ; generate start bit
0075 020F           00277           movf    slave,w     ; get slave address
0076 0030           00278           movwf   txbuf       ; into transmit buffer
0077 0952           00279           call    TX          ; and send it
0078 0217           00280           movf    addr1,w     ; get word 1 address
0079 0030           00281           movwf   txbuf       ; into transmit buffer
007A 0952           00282           call    TX          ; and send it
007B 020C           00283           movf    addr,w      ; get word 0 address
007C 0030           00284           movwf   txbuf       ; into transmit buffer
007D 0952           00285           call    TX          ; and send it
007E 090E           00286           call    BSTART      ; generate start bit
007F 0CA1           00287           movlw   b'10100001' ; get slave address and read mode
0080 0030           00288           movwf   txbuf       ; into transmit buffer
0081 0952           00289           call    TX          ; and transmit it
                    00290                               ;
0082 095F           00291 rbyte     call    RX          ; read 1 byte from device
0083 02F2           00292           decfsz  bcount, F   ; are all 8 bytes read?
0084 0A8C           00293           goto    lowack      ; no, send low ack and do another
0085 05CA           00294           bsf     eeprom,do   ; yes, send high ack bit
0086 0936           00295           call    BITOUT      ; to stop tranxmission
0087 0923           00296           call    BSTOP       ; and send a stop bit
                    00297
0088 0CFF           00298           movlw   .255        ; long delay for scope
0089 0035           00299           movwf   loops       ; trigger purposes only
008A 0901           00300           call    WAIT
008B 0A6D           00301           goto    READ        ; start all over
                    00302
008C 04CA           00303 lowack    bcf     eeprom,do   ; send low ack bit
008D 0936           00304           call    BITOUT      ; to continue transmission
008E 0A82           00305           goto    rbyte       ; and read another byte
                    00306
                    00307 ;
                    00308     END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXX- ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   144
```


© 1997 Microchip Technology Inc.

```
Program Memory Words Free:   368
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX F: 64KSETHE.ASM

```
MPASM 01.40 Released          64KSETHE.ASM   1-16-1997  16:33:09          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54
                     00002 ;*****************************************************************
                     00003 ;       Program to set the high endurance block on the 64K
                     00004 ;       device (24LC65).      (122 bytes)
                     00005 ;
                     00006 ;       The 24LC65 is a 64K device, 4K of which is error
                     00007 ;       corrected to increase the endurance of that portion
                     00008 ;       of the array.  The location of this high endurance
                     00009 ;       'block' can be set by the user.  When the device
                     00010 ;       comes from the factory, the high endurance block
                     00011 ;       will be set as the uppermost block in the array.
                     00012 ;
                     00013 ;       This program sets the high endurance block as the
                     00014 ;       first block in the array (address range 00h to 1FFh).
                     00015 ;
                     00016 ;       The high endurance block is set by sending the following
                     00017 ;       sequence to the device:
                     00018 ;
                     00019 ;       SB 10100000 1XXAAAAX XXXXXXXX 00XX0000 STB
                     00020 ;
                     00021 ;       Where SB=start bit
                     00022 ;              1=data high
                     00023 ;              0=data low
                     00024 ;              X=don't care
                     00025 ;           AAAA=4 bit high endurance block address
                     00026 ;            STB=stop bit
                     00027 ;
                     00028 ;       As an option, the user may connect a LED to pin 18
                     00029 ;       on the PIC (use about a 1K resistor in series) as an
                     00030 ;       acknowledge fail indicator.  This LED will come on if
                     00031 ;       the serial EE fails to acknowledge correctly.
                     00032 ;
                     00033 ;       Timing is based on using the PIC in 'XT' mode
                     00034 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00035 ;       will be approximately 60 Khz for this setup.
                     00036 ;
                     00037 ;        PIC to Serial EE Connections:
                     00038 ;
                     00039 ;           PIC          Serial EE
                     00040 ;       ------------    ------------
                     00041 ;       Pin 12 (RB6) -->  SCLK
                     00042 ;       Pin 13 (RB7) -->  SDATA
                     00043 ;
                     00044 ;       PIN 18 (RA1) -->  Acknowledge fail LED  (Optional);
                     00045 ;
                     00046 ;       Program:          64KSETHE.ASM
                     00047 ;       Revision Date:
                     00048 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00049 ;
                     00050 ;*****************************************************************
                     00051 ;          Register Definitions
                     00052 ;*****************************************************************
  00000005           00053 port_a  equ     5h      ; port 5 (port_a) used for LEDs
  00000006           00054 port_b  equ     6h      ; port 6 (port b) used for data and
```


© 1997 Microchip Technology Inc.

```
                    00055                           ; clock lines
  0000000A          00056 eeprom  equ     0ah       ; bit buffer
  0000000B          00057 bycnt   equ     0bh       ; byte counter for read mode
  0000000C          00058 addr    equ     0ch       ; word 0 address counter
  0000000D          00059 datai   equ     0dh       ; data input register
  0000000E          00060 datao   equ     0eh       ; data output register
  0000000F          00061 slave   equ     0fh       ; device address (1010xxx0)
  00000010          00062 txbuf   equ     10h       ; transmit buffer
  00000011          00063 count   equ     11h       ; bit counter
  00000012          00064 bcount  equ     12h       ; byte counter
  00000015          00065 loops   equ     15h       ; delay loop counter
  00000016          00066 loops2  equ     16h       ; delay loop counter
  00000017          00067 addr1   equ     17h       ; word 1 address counter
  00000018          00068 he_blk  equ     18h       ; high endurance block address
                    00069 ;*****************************************************
                    00070 ;             Bit Definitions
                    00071 ;*****************************************************
  00000007          00072 di      equ     7         ; eeprom input bit
  00000006          00073 do      equ     6         ; eeprom output bit
  00000007          00074 sdata   equ     7         ; serial EE data line (port_b,pin 13)
  00000006          00075 sclk    equ     6         ; serail EE clock line (port_b,pin 12)
  00000001          00076 ackf    equ     1         ; acknowledge fail LED (port_a,pin 18)
                    00077 ;*****************************************************
                    00078 ;
                    00079 ;
01FF                00080         org     01ffh     ; set reset vector
01FF 0A5E           00081         goto    PWRUP
0000                00082         org     000h      ;
0000 0A5E           00083         goto    PWRUP
                    00084 ;
                    00085 ;*****************************************************
                    00086 ;       DELAY ROUTINE
                    00087 ;       This routine takes the value in 'loops'
                    00088 ;       and multiplies it times 1 millisecond to
                    00089 ;       determine delay time.
                    00090 ;*****************************************************
0001                00091 WAIT
                    00092 ;
0001 0C6E           00093 top       movlw   .110            ; timing adjustment variable
0002 0036           00094           movwf   loops2
0003 0000           00095 top2      nop                     ; sit and wait
0004 0000           00096           nop
0005 0000           00097           nop
0006 0000           00098           nop
0007 0000           00099           nop
0008 0000           00100           nop
0009 02F6           00101           decfsz  loops2, F       ; inner loops complete?
000A 0A03           00102           goto    top2            ; no, go again
                    00103                                   ;
000B 02F5           00104           decfsz  loops, F        ; outer loops complete?
000C 0A01           00105           goto    top             ; no, go again
000D 0800           00106           retlw   0               ; yes, return from sub
                    00107 ;
                    00108 ;*****************************************************
                    00109 ;       Start Bit Subroutine
                    00110 ;       this routine generates a start bit
                    00111 ;       (Low going data line while clock is high)
                    00112 ;*****************************************************
                    00113 ;
000E                00114 BSTART
000E 05E6           00115           bsf     port_b,sdata    ; make sure data is high
000F 0C3F           00116           movlw   b'00111111'
0010 0006           00117           tris    port_b          ; set data and clock lines for output
0011 04C6           00118           bcf     port_b,sclk     ; make sure clock is low
0012 0000           00119           nop
0013 05C6           00120           bsf     port_b,sclk     ; set clock high
```


© 1997 Microchip Technology Inc.

```
0014 0000           00121          nop
0015 0000           00122          nop
0016 0000           00123          nop
0017 0000           00124          nop
0018 0000           00125          nop
0019 04E6           00126          bcf     port_b,sdata    ; data line goes low during
                    00127                                  ; high clock for start bit
001A 0000           00128          nop
001B 0000           00129          nop
001C 0000           00130          nop
001D 0000           00131          nop
001E 0000           00132          nop                    ; timing adjustment
001F 04C6           00133          bcf     port_b,sclk    ; start clock train
0020 0000           00134          nop
0021 0000           00135          nop
0022 0800           00136          retlw   0
                    00137 ;
                    00138 ;*********************************************************
                    00139 ;       Stop Bit Subroutine
                    00140 ;       This routine generates a stop bit
                    00141 ;       (High going data line while clock is high)
                    00142 ;*********************************************************
0023                00143 BSTOP
0023 0C3F           00144          movlw   b'00111111'    ;
0024 0006           00145          tris    port_b         ; set data/clock lines as outputs
0025 04E6           00146          bcf     port_b,sdata   ; make sure data line is low
0026 0000           00147          nop
0027 0000           00148          nop
0028 0000           00149          nop
0029 05C6           00150          bsf     port_b,sclk    ; set clock high
002A 0000           00151          nop
002B 0000           00152          nop
002C 0000           00153          nop
002D 05E6           00154          bsf     port_b,sdata   ; data goes high while clock high
                    00155                                  ; for stop bit
002E 0000           00156          nop
002F 0000           00157          nop
0030 04C6           00158          bcf     port_b,sclk    ; set clock low again
0031 0000           00159          nop
0032 0000           00160          nop
0033 0000           00161          nop
0034 0800           00162          retlw   0
                    00163 ;
                    00164 ;*********************************************************
                    00165 ;       BITOUT routine takes one bit of data in 'do' and
                    00166 ;       transmits it to the serial EE device
                    00167 ;*********************************************************
0035                00168 BITOUT
0035 0C3F           00169          movlw   b'00111111'    ; set data,clock as outputs
0036 0006           00170          tris    port_b
0037 07CA           00171          btfss   eeprom,do      ; check for state of data bit to xmit
0038 0A3B           00172          goto    bitlow         ;
0039 05E6           00173          bsf     port_b,sdata   ; set data line high
003A 0A3C           00174          goto    clkout         ; go toggle the clock
                    00175
003B 04E6           00176 bitlow   bcf     port_b,sdata   ; output a low bit
003C 05C6           00177 clkout   bsf     port_b,sclk    ; set clock line high
003D 0000           00178          nop
003E 0000           00179          nop
003F 0000           00180          nop
0040 0000           00181          nop
0041 04C6           00182          bcf     port_b,sclk    ; return clock line low
0042 0800           00183          retlw   0
                    00184 ;
                    00185 ;*********************************************************
                    00186 ;       BITIN routine reads one bit of data from the
```


© 1997 Microchip Technology Inc.

```
                        00187 ;       serial EE device and stores it in 'di'
                        00188 ;*****************************************************************
0043                    00189 BITIN
0043 05EA               00190         bsf     eeprom,di       ; assume input bit is high
0044 0CBF               00191         movlw   b'10111111'     ; make sdata an input line
0045 0006               00192         tris    port_b
0046 05E6               00193         bsf     port_b,sdata    ; set sdata line for input
0047 05C6               00194         bsf     port_b,sclk     ; set clock line high
0048 0000               00195         nop                     ; just sit here a sec
0049 0000               00196         nop
004A 0000               00197         nop
004B 0000               00198         nop
004C 0000               00199         nop                     ;
004D 07E6               00200         btfss   port_b,sdata    ; read the data bit
004E 04EA               00201         bcf     eeprom,di       ; input bit was low
004F 04C6               00202         bcf     port_b,sclk     ; set clock line low
                        00203                                 ;
0050 0800               00204         retlw   0               ;
                        00205 ;
                        00206 ;*****************************************************************
                        00207 ;       Transmit Data Subroutine
                        00208 ;       This routine takes the byte of data stored in the
                        00209 ;       'datao' register and transmits it to the serial EE device.
                        00210 ;       It will then send 1 more clock to the serial EE for the
                        00211 ;       acknowledge bit.  If the ack bit from the part was low
                        00212 ;       then the transmission was sucessful.  If it is high, then
                        00213 ;       the device did not send a proper ack bit and the ack
                        00214 ;       fail LED will be turned on.
                        00215 ;*****************************************************************
0051                    00216 TX
0051 0C08               00217         movlw   .8
0052 0031               00218         movwf   count           ; set the #bits to 8
                        00219         ;
0053                    00220 TXLP
0053 04CA               00221         bcf     eeprom,do       ; assume bit out is low
0054 06F0               00222         btfsc   txbuf,7         ; is bit out really low?
0055 05CA               00223         bsf     eeprom,do       ; otherwise data bit =1
0056 0935               00224         call    BITOUT          ; serial data out
0057 0370               00225         rlf     txbuf, F        ; rotate txbuf left
0058 02F1               00226         decfsz  count, F        ; 8 bits done?
0059 0A53               00227         goto    TXLP            ; no - go again
005A 0943               00228         call    BITIN           ; read ack bit
005B 06EA               00229         btfsc   eeprom,di       ; check ack bit
005C 0525               00230         bsf     port_a,ackf     ; set acknowledge fail LED if the
                        00231         ;
005D 0800               00232         retlw   0
                        00233 ;
                        00234 ;*****************************************************************
                        00235 ;       Power up routine
                        00236 ;       This is the program entry point, which in this case simply
                        00237 ;       sets the port_a I/O lines and directs control to the
                        00238 ;       byte write routine.
                        00239 ;*****************************************************************
005E                    00240 PWRUP
005E 0C00               00241           movlw  b'00000000'
005F 0005               00242           tris   port_a           ; set port A as all output
0060 0065               00243           clrf   port_a           ; all output lines low
0061 0A62               00244           goto   setheblk         ; set go set the high endurance block
                        00245
                        00246 ;*****************************************************************
                        00247 ;       Set High Endurance Block Routine
                        00248 ;       This routine sets the high endurance block to the first
                        00249 ;       block in the array and then delays about a half second.  This
                        00250 ;       routine will repeat forever.
                        00251 ;*****************************************************************
                        00252 ;
```


© 1997 Microchip Technology Inc.

```
0062                    00253 setheblk
                        00254 ;
0062 0065               00255         clrf    port_a          ; clear all LEDs
0063 002F               00256         movwf   slave
0064 0C00               00257         movlw   .0
0065 0038               00258         movwf   he_blk          ; set the endurance block as first
                        00259                                 ; block in array (block 0)
0066 0378               00260         rlf     he_blk, F       ; rotate starting block left 1 bit
                        00261                                 ; to arrange it correctly
0067 05F8               00262         bsf     he_blk,7        ; set msb bit in block address byte
0068 090E               00263         call    BSTART          ; generate start bit
0069 0CA0               00264         movlw   b'10100000'     ; get slave address and write mode
006A 0030               00265         movwf   txbuf           ; into transmit buffer
006B 0951               00266         call    TX              ; and send it
006C 0218               00267         movf    he_blk,w        ; get the block address byte
006D 0030               00268         movwf   txbuf           ; into transmit buffer
006E 0951               00269         call    TX              ; and send it
006F 0070               00270         clrf    txbuf           ; clear buffer
0070 0951               00271         call    TX              ; send 8 don't care bits
0071 0070               00272         clrf    txbuf
0072 0951               00273         call    TX              ; send config byte(all 0's)
0073 0923               00274         call    BSTOP           ; send stop bit
                        00275
                        00276
0074 0CFF               00277         movlw   .255            ; long delay
0075 0035               00278         movwf   loops
0076 0901               00279         call    WAIT
0077 0CFF               00280         movlw   .255
0078 0035               00281         movwf   loops
0079 0901               00282         call    WAIT
                        00283
007A 0A62               00284         goto    setheblk        ; go again
                        00285 ;
                        00286 ;
                        00287     END

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX-----
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   124
Program Memory Words Free:   388
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX G: 64KSETS.ASM

```
MPASM 01.40 Released          64KSETS.ASM   1-16-1997  16:33:32          PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54
                     00002 ;*****************************************************************
                     00003 ;       Program to set the security option (write protect)
                     00004 ;       on the 64K device (24LC65).    (125 bytes)
                     00005 ;
                     00006 ;       The security option (or write protect option) allows
                     00007 ;       the user to write protect any number of consecutive
                     00008 ;       4K blocks in the device.
                     00009 ;
                     00010 ;       This program sets the security option to protect the
                     00011 ;       lower 4 blocks (2K bytes) in the array.
                     00012 ;
                     00013 ;       **********  CAUTIONS TO THE USER !!!! **********
                     00014 ;
                     00015 ;       1) THE SECURITY OPTION CAN ONLY BE SET ONCE!!
                     00016 ;       2) THE HIGH ENDURANCE BLOCK CANNOT BE CHANGED
                     00017 ;          AFTER THE SECURITY OPTION IS SET.
                     00018 ;       3) THE HIGH ENDURANCE BLOCK CANNOT BE PROTECTED.
                     00019 ;
                     00020 ;       The security option is set by sending the following
                     00021 ;       sequence to the device:
                     00022 ;
                     00023 ;       SB 10100000 1XXAAAAX XXXXXXXX 10XXNNNN STB
                     00024 ;
                     00025 ;       Where SB=start bit
                     00026 ;              1=data high
                     00027 ;              0=data low
                     00028 ;              X=don't care
                     00029 ;           AAAA=4 bit number to indicate first secure block (0-15)
                     00030 ;           NNNN=4 bit number to indicate how many secure blocks(1-15)
                     00031 ;            STB=stop bit
                     00032 ;
                     00033 ;       As an option, the user may connect a LED to pin 18
                     00034 ;       on the PIC (use about a 1K resistor in series) as an
                     00035 ;       acknowledge fail indicator.  This LED will come on if
                     00036 ;       the serial EE fails to acknowledge correctly.
                     00037 ;
                     00038 ;       Timing is based on using the PIC in 'XT' mode
                     00039 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00040 ;       will be approximately 60 Khz for this setup.
                     00041 ;
                     00042 ;        PIC to Serial EE Connections:
                     00043 ;
                     00044 ;           PIC          Serial EE
                     00045 ;       ------------    ------------
                     00046 ;       Pin 12 (RB6) -->  SCLK
                     00047 ;       Pin 13 (RB7) -->  SDATA
                     00048 ;
                     00049 ;       PIN 18 (RA1) -->  Acknowledge fail LED  (Optional);
                     00050 ;
                     00051 ;       Program:          64KSETS.ASM
                     00052 ;       Revision Date:
                     00053 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00054 ;
```


© 1997 Microchip Technology Inc.

```
                    00055 ;*******************************************************
                    00056 ;           Register Definitions
                    00057 ;*******************************************************
  00000005          00058 port_a      equ     5h      ; port 5 (port_a) used for LEDs
  00000006          00059 port_b      equ     6h      ; port 6 (port b) used for data and
                    00060                             ; clock lines
  0000000A          00061 eeprom      equ     0ah     ; bit buffer
  0000000B          00062 bycnt       equ     0bh     ; byte counter for read mode
  0000000C          00063 addr        equ     0ch     ; word 0 address counter
  0000000D          00064 datai       equ     0dh     ; data input register
  0000000E          00065 datao       equ     0eh     ; data output register
  0000000F          00066 slave       equ     0fh     ; device address (1010xxx0)
  00000010          00067 txbuf       equ     10h     ; transmit buffer
  00000011          00068 count       equ     11h     ; bit counter
  00000012          00069 bcount      equ     12h     ; byte counter
  00000015          00070 loops       equ     15h     ; delay loop counter
  00000016          00071 loops2      equ     16h     ; delay loop counter
  00000017          00072 addr1       equ     17h     ; word 1 address counter
  00000018          00073 strt_blk    equ     18h     ; starting block number for secure portion
  00000019          00074 sec_blks    equ     19h     ; number of secure blocks
                    00075 ;*******************************************************
                    00076 ;           Bit Definitions
                    00077 ;*******************************************************
  00000007          00078 di      equ     7       ; eeprom input bit
  00000006          00079 do      equ     6       ; eeprom output bit
  00000007          00080 sdata   equ     7       ; serial EE data line (port_b,pin 13)
  00000006          00081 sclk    equ     6       ; serail EE clock line (port_b,pin 12)
  00000001          00082 ackf    equ     1       ; acknowledge fail LED (port_a,pin 18)
                    00083 ;*******************************************************
                    00084 ;
                    00085 ;
01FF                00086         org     01ffh   ; set reset vector
01FF 0A5E           00087         goto    PWRUP
0000                00088         org     000h    ;
0000 0A5E           00089         goto    PWRUP
                    00090 ;
                    00091 ;*******************************************************
                    00092 ;       DELAY ROUTINE
                    00093 ;       This routine takes the value in 'loops'
                    00094 ;       and multiplies it times 1 millisecond to
                    00095 ;       determine delay time.
                    00096 ;*******************************************************
0001                00097 WAIT
                    00098 ;
0001 0C6E           00099 top       movlw   .110            ; timing adjustment variable
0002 0036           00100           movwf   loops2
0003 0000           00101 top2      nop                     ; sit and wait
0004 0000           00102           nop
0005 0000           00103           nop
0006 0000           00104           nop
0007 0000           00105           nop
0008 0000           00106           nop
0009 02F6           00107           decfsz  loops2, F       ; inner loops complete?
000A 0A03           00108           goto    top2            ; no, go again
                    00109                                   ;
000B 02F5           00110           decfsz  loops, F        ; outer loops complete?
000C 0A01           00111           goto    top             ; no, go again
000D 0800           00112           retlw   0               ; yes, return from sub
                    00113 ;
                    00114 ;*******************************************************
                    00115 ;       Start Bit Subroutine
                    00116 ;       this routine generates a start bit
                    00117 ;       (Low going data line while clock is high)
                    00118 ;*******************************************************
                    00119 ;
000E                00120 BSTART
```


© 1997 Microchip Technology Inc.

```
000E 05E6          00121          bsf     port_b,sdata    ; make sure data is high
000F 0C3F          00122          movlw   b'00111111'
0010 0006          00123          tris    port_b          ; set data and clock lines for output
0011 04C6          00124          bcf     port_b,sclk     ; make sure clock is low
0012 0000          00125          nop
0013 05C6          00126          bsf     port_b,sclk     ; set clock high
0014 0000          00127          nop
0015 0000          00128          nop
0016 0000          00129          nop
0017 0000          00130          nop
0018 0000          00131          nop
0019 04E6          00132          bcf     port_b,sdata    ; data line goes low during
                   00133                                  ; high clock for start bit
001A 0000          00134          nop
001B 0000          00135          nop
001C 0000          00136          nop
001D 0000          00137          nop
001E 0000          00138          nop                     ; timing adjustment
001F 04C6          00139          bcf     port_b,sclk     ; start clock train
0020 0000          00140          nop
0021 0000          00141          nop
0022 0800          00142          retlw   0
                   00143 ;
                   00144 ;*****************************************************************
                   00145 ;       Stop Bit Subroutine
                   00146 ;       This routine generates a stop bit
                   00147 ;       (High going data line while clock is high)
                   00148 ;*****************************************************************
0023               00149 BSTOP
0023 0C3F          00150          movlw   b'00111111'     ;
0024 0006          00151          tris    port_b          ; set data/clock lines as outputs
0025 04E6          00152          bcf     port_b,sdata    ; make sure data line is low
0026 0000          00153          nop
0027 0000          00154          nop
0028 0000          00155          nop
0029 05C6          00156          bsf     port_b,sclk     ; set clock high
002A 0000          00157          nop
002B 0000          00158          nop
002C 0000          00159          nop
002D 05E6          00160          bsf     port_b,sdata    ; data goes high while clock high
                   00161                                  ; for stop bit
002E 0000          00162          nop
002F 0000          00163          nop
0030 04C6          00164          bcf     port_b,sclk     ; set clock low again
0031 0000          00165          nop
0032 0000          00166          nop
0033 0000          00167          nop
0034 0800          00168          retlw   0
                   00169 ;
                   00170 ;*****************************************************************
                   00171 ;       BITOUT routine takes one bit of data in 'do' and
                   00172 ;       transmits it to the serial EE device
                   00173 ;*****************************************************************
0035               00174 BITOUT
0035 0C3F          00175          movlw   b'00111111'     ; set data,clock as outputs
0036 0006          00176          tris    port_b
0037 07CA          00177          btfss   eeprom,do       ; check for state of data bit to xmit
0038 0A3B          00178          goto    bitlow          ;
0039 05E6          00179          bsf     port_b,sdata    ; set data line high
003A 0A3C          00180          goto    clkout          ; go toggle the clock
                   00181
003B 04E6          00182 bitlow   bcf     port_b,sdata    ; output a low bit
003C 05C6          00183 clkout   bsf     port_b,sclk     ; set clock line high
003D 0000          00184          nop
003E 0000          00185          nop
003F 0000          00186          nop
```

7

© 1997 Microchip Technology Inc.

```
0040 0000           00187          nop
0041 04C6           00188          bcf     port_b,sclk     ; return clock line low
0042 0800           00189          retlw   0
                    00190 ;
                    00191 ;*****************************************************************
                    00192 ;       BITIN routine reads one bit of data from the
                    00193 ;       serial EE device and stores it in 'di'
                    00194 ;*****************************************************************
0043                00195 BITIN
0043 05EA           00196          bsf     eeprom,di       ; assume input bit is high
0044 0CBF           00197          movlw   b'10111111'     ; make sdata an input line
0045 0006           00198          tris    port_b
0046 05E6           00199          bsf     port_b,sdata    ; set sdata line for input
0047 05C6           00200          bsf     port_b,sclk     ; set clock line high
0048 0000           00201          nop                     ; just sit here a sec
0049 0000           00202          nop
004A 0000           00203          nop
004B 0000           00204          nop
004C 0000           00205          nop                     ;
004D 07E6           00206          btfss   port_b,sdata    ; read the data bit
004E 04EA           00207          bcf     eeprom,di       ; input bit was low
004F 04C6           00208          bcf     port_b,sclk     ; set clock line low
                    00209                                  ;
0050 0800           00210          retlw   0               ;
                    00211 ;
                    00212 ;*****************************************************************
                    00213 ;       Transmit Data Subroutine
                    00214 ;       This routine takes the byte of data stored in the
                    00215 ;       'datao' register and transmits it to the serial EE device.
                    00216 ;       It will then send 1 more clock to the serial EE for the
                    00217 ;       acknowledge bit.  If the ack bit from the part was low
                    00218 ;       then the transmission was sucessful.  If it is high, then
                    00219 ;       the device did not send a proper ack bit and the ack
                    00220 ;       fail LED will be turned on.
                    00221 ;*****************************************************************
0051                00222 TX
0051 0C08           00223          movlw   .8
0052 0031           00224          movwf   count           ; set the #bits to 8
                    00225          ;
0053                00226 TXLP
0053 04CA           00227          bcf     eeprom,do       ; assume bit out is low
0054 06F0           00228          btfsc   txbuf,7         ; is bit out really low?
0055 05CA           00229          bsf     eeprom,do       ; otherwise data bit =1
0056 0935           00230          call    BITOUT          ; serial data out
0057 0370           00231          rlf     txbuf, F        ; rotate txbuf left
0058 02F1           00232          decfsz  count, F        ; 8 bits done?
0059 0A53           00233          goto    TXLP            ; no - go again
005A 0943           00234          call    BITIN           ; read ack bit
005B 06EA           00235          btfsc   eeprom,di       ; check ack bit
005C 0525           00236          bsf     port_a,ackf     ; set acknowledge fail LED if the
                    00237          ;
005D 0800           00238          retlw   0
                    00239 ;
                    00240 ;*****************************************************************
                    00241 ;       Power up routine
                    00242 ;       This is the program entry point, which in this case simply
                    00243 ;       sets the port_a I/O lines and directs control to the
                    00244 ;       byte write routine.
                    00245 ;*****************************************************************
005E                00246 PWRUP
005E 0C00           00247           movlw  b'00000000'
005F 0005           00248           tris   port_a           ; set port A as all output
0060 0065           00249           clrf   port_a           ; all output lines low
0061 0A62           00250           goto   set_sec          ; set go set the high endurance block
                    00251
                    00252 ;*****************************************************************
```


© 1997 Microchip Technology Inc.

```
                    00253 ;       Set Security Routine
                    00254 ;       This routine sets the secure portion of the array to the
                    00255 ;       lower 4 blocks (2K bytes) in the array and then delays
                    00256 ;       about a half second.  This routine will repeat forever.
                    00257 ;*****************************************************************
                    00258 ;
0062                00259 set_sec
                    00260 ;
0062 0065           00261         clrf    port_a          ; clear all LEDs
0063 0C00           00262         movlw   .0
0064 0038           00263         movwf   strt_blk        ; set the first protected block as
                    00264                                 ; block 0
0065 0C04           00265         movlw   .4
0066 0039           00266         movwf   sec_blks        ; set the number of protected blocks
                    00267                                 ; as 4 (2048 bytes)
0067 0378           00268         rlf     strt_blk, F     ; rotate starting block left 1 bit
                    00269                                 ; to arrange it correctly
0068 05F8           00270         bsf     strt_blk,7      ; set msb bit in starting block address
0069 05F9           00271         bsf     sec_blks,7      ; set msb bit block count byte
006A 090E           00272         call    BSTART          ; generate start bit
006B 0CA0           00273         movlw   b'10100000'     ; get slave address and write mode
006C 0030           00274         movwf   txbuf           ; into transmit buffer
006D 0951           00275         call    TX              ; and send it
                    00276
006E 0218           00277         movf    strt_blk,w      ; get the starting block address
006F 0030           00278         movwf   txbuf           ; into transmit buffer
0070 0951           00279         call    TX              ; and send it
0071 0070           00280         clrf    txbuf           ; clear buffer
0072 0951           00281         call    TX              ; send 8 don't care bits
                    00282
0073 0219           00283         movf    sec_blks,w      ; get secure blocks byte
0074 0030           00284         movwf   txbuf           ; into transmit buffer
0075 0951           00285         call    TX              ; and send it
0076 0923           00286         call    BSTOP           ; send stop bit
                    00287
                    00288
0077 0CFF           00289         movlw   .255            ; long delay
0078 0035           00290         movwf   loops
0079 0901           00291         call    WAIT
007A 0CFF           00292         movlw   .255
007B 0035           00293         movwf   loops
007C 0901           00294         call    WAIT
                    00295
007D 0A62           00296         goto    set_sec         ; go again
                    00297 ;
                    00298 ;
                    00299     END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXX--
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   127
Program Memory Words Free:   385
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN559

Optimizing Serial Bus Operations with Proper Write Cycle Times

Author: *Lenny French*
Microchip Technology Inc.

SERIAL EEPROM WRITE TIME REQUIREMENTS

Elements of the Write Cycle Time

The total write operation time for a Serial EEPROM is determined by three main elements:

- Number of bytes to load for each write operation
- Bus clock speed at which the write operation is loaded
- Fixed internal write cycle timer required for the programming operation

The load component of the write command consists of the control byte, address, and the data of up to 16 bytes. The time required to load the operation depends on the number of bytes to load at one time and the bus clock speed. After this load is complete, the part commences the internally controlled write cycle and the bus and system are free to perform other tasks. The internal write cycle timer is a fixed time delay which is required to program the EEPROM memory cells. Table 1 gives examples of total write time for 1 and 16 bytes for various parts at fast and normal clock speeds.

- **Load time (time at bus free)** is the time the part is being loaded with the instruction, address, and data. The bus is free after this time interval, and the part commences the internally controlled write cycle sequence.
- **Write timer (worse case)** indicates the time the part is in the internally controlled write cycle allowing for the maximum specified datasheet requirements.
- **Write timer (typical)** indicates the time the part is in the internally controlled write cycle assuming nominal conditions and utilizing write cycle polling.
- **Total write** time is the combined load time and typical internal write cycle time.

MINIMIZING SERIAL BUS COMMUNICATION TIME IN A SYSTEM

Utilizing the Page Write Option

The original Microchip Serial EEPROM products, though utilizing a page buffer, only write bytes sequentially. This means the time required to write 8 bytes is 8 ms, worse case. The new 24LCXX products incorporate a page mode that allows simultaneous writes of up to 16 bytes. This allows the programming of up to 16 bytes in one write cycle (10 ms) compared to 16 write cycles (16 ms) for the original 24CXX products. Microchip uses an 8 byte page in the 24LC01 and 24LC02, and a 16 byte page in the 24LC04 and higher densities. What this means to a system designer is a write of 8 bytes in a 24LC01 or 24LC02 would take 10 ms, worse

7

TABLE 1: WRITE OPERATION TIME COMPARISON

Product	Page Width	# Bytes to load	Speed (KHz)	Load time (ms) time at bus free	Write timer (ms) Worse case	Write timer (ms) Typical (25C)	Typical total write time (ms)
24C01A	2	1	100	0.28	1	0.4	0.68
		8 (4 x 2)	100	1.12	1	0.4	4.32
24LC01	8	1	100	0.28	10	3	3.28
		8	100	0.91	10	3	3.91
		1	400	0.07	10	3	3.07
		8	400	0.23	10	3	3.23
24C04A	8	1	100	0.28	1	0.4	0.68
		16 (2 x 8)	100	1.82	1	0.4	8.22
24LC04	16	1	100	0.28	10	3	3.28
		16	100	1.63	10	3	4.63
		1	400	0.07	10	3	3.07
		16	400	0.41	10	3	3.41

© 1997 Microchip Technology Inc.

case, versus 8 ms, worse case, in either a 24C01A or 24C02A. Loading the entire memory of an original 24C04A takes 512 ms, but the same operation on a new 24LC04 is reduced to only 320 ms, assuming worse case. As Table 1 shows, the difference is even greater using typical numbers.

The 93LCXX products do not utilize a page mode. However, in 16 bit mode, all 16 bits are written simultaneously. The original 93CXX products write the 16 bits in 2 write cycles, so the write cycle time increase of the new products is only 5 X (3.5 X typical).

Utilizing Write Cycle Polling

One powerful method of increasing programming efficiency is by periodically polling the part to determine if the write cycle has completed. To poll the 24LCXX series products, a control byte is sent and the acknowledge bit from the part is read. If the part acknowledges (pulls SDA low), it is ready to accept a new command. The part will not acknowledge while in the internally timed write cycle.

To poll the 93LCXX series products, the chip select is pulled high after the write cycle commences, and the data out line is read for the ready/busy status. If the part is still busy, it will pull the data line low. When the internally timed write cycle is complete, the part will pull the data line high, indicating it is ready for a new command.

Serial EEPROM System Optimization

Serial EEPROMs are used in systems for two purposes: storing data and reading back data. Read operations are at full clock speed, so the only methods for optimization are to run the clock at maximum frequency and to utilize sequential read whenever possible. Sequential read allows a continuous output data stream on one command.

The write operation, with its internal write timer component, needs special consideration when designing the control software. Efficient operation can be accomplished using both the page mode and write cycle polling. The following example shows a typical fetch-store operation in a system and how optimization can be incorporated. In the example system, the microcontroller must fetch bytes of data from a sensor and send the bytes to a EEPROM for storage. Two cases are shown in Figure 2: case 1 uses a 24C01A with no optimization; case 2 uses a 24LC01B with the available page mode and write cycle polling.

By utilizing the available page write mode and by polling for the write cycle completion, nearly four times as many bytes can be initially loaded to the serial EEPROM in the same time interval. Continuous operation for the optimized case 2 takes only 0.49 ms per byte compared to 1.28 ms per byte for the non-optimized case 1.

© 1997 Microchip Technology Inc.

FIGURE 1: CASE 1 (NO OPTIMIZATION)

Case 1 (No Optimization):		**Case 2 (using page mode and polling):**	
Operation	**Serial Bus Time**	**Operation**	**Serial Bus Time**
fetch byte		fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		fetch byte	
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		fetch byte	
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		load 8 bytes to serial	0.91 ms
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		fetch byte	
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		fetch byte	
wait for write timer	1 ms	fetch byte	
load to serial	0.28 ms	fetch byte	
fetch byte		poll for write timer	<3 ms typical
wait for write timer	1 ms	load 8 bytes to serial	0.91 ms
load to serial	0.28 ms		
fetch byte			
wait for write timer	1 ms		
load to serial	0.28 ms		
First **8 bytes** loaded in 9.24 ms		First **16 bytes** loaded in 4.82 ms	

7

© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Using the 93LC56 and 93LC66

Author: *Bruce Negley*
Microchip Technology Inc.

INTRODUCTION

The Microchip Technology Inc. 93LC56/66 are low-power 3-wire non-volatile memories and are suitable for many embedded system code and data storage applications. These devices are easily interfaced to most microcontrollers in today's market place, but Microchip's 8-bit RISC series PIC16CXXX offers the best code density of any microcontroller on the market today. Using the PIC16C54, the assembly programs contained in this application note have been fully tested and provide the correct timing and 3-wire sequences to fully operate the 93LC56/66 in a PIC16CXXX-based embedded application. The PIC16C54 was clocked at a 10MHz frequency. This application note is intended to provide the engineer with readily available stand-alone code modules to accomplish all of the necessary functions to utilize these devices in a low power application using the efficient PIC16C54 microcontroller.

The 93 series of devices have essentially four I/O pins:

CS	Chip Select
CLK	Clock
DI	Data In
DO	Data Out

This series of devices use a series of commands to accomplish the normal memory functions. These are READ, WRITE, EWEN, ERASE, ERAL, WRAL, EWDS. For a more detailed discussion of the function of these devices reference the appropriate data sheet and AN536, also published by Microchip Technology.

The following programs are included in this application note and are fully functional stand-alone modules. They are intended for use by those who are not already familiar with interfacing a PIC16CXXX microcontroller to a 93 series device. For those with more experience, please refer to application note AN530.

3-Wire Byte Read Program

- Start Bit Routine
- Receive Data Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Read Routine

3-Wire Byte Write Program

- Delay Routine
- Start Bit Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Erase/Write Enable Routine (EWEN)
- Byte Write Routine
- Erase/Write Disable Routine (EWDS)

3-Wire Byte Write with Data Polling Program

- Data Polling Delay Routine
- Start Bit Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Erase/Write Enable Routine
- Write Routine
- Erase/Write Disable Routine (EWDS)

3-Wire Sequential Read Program

- Delay Routine
- Start Bit Routine
- Bit In Routine
- Receive Data Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Read Routine

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: 3WBYTER.ASM

```
MPASM 01.40 Released          3WBYTER.ASM   1-16-1997  16:38:43         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54
                     00002 ;*****************************************************************
                     00003 ;       3-Wire Byte Read Program (80 bytes)
                     00004 ;
                     00005 ;       This program demonstrates how to interface a
                     00006 ;       Microchip PIC 16C54 to a 93LC56 or 93LC66 Serial EE
                     00007 ;       device.  This program will read 8 consecutive addresses
                     00008 ;       in the 'random read' mode.  This means that the opcode
                     00009 ;       and address for each byte will be sent to the device.
                     00010 ;       This program will repeat forever.
                     00011 ;
                     00012 ;       Another, more efficient method of reading consecutive
                     00013 ;       addresses is called the 'sequential read' mode.  This
                     00014 ;       involves sending the opcode and address for the first
                     00015 ;       byte to read, then continuing to provide clocks for the
                     00016 ;       next addresses. The device will automatically increment
                     00017 ;       the address.  An example of the seqential read mode is
                     00018 ;       provided in the '3wseqr.asm' file.
                     00019 ;
                     00020 ;       This program communicates to the serial EE in the
                     00021 ;       x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
                     00022 ;       ON THE DEVICE TO Vcc.
                     00023 ;
                     00024 ;       Timing is based on using the PIC in 'XT' mode
                     00025 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00026 ;       will be approximately 50 Khz for this setup.
                     00027 ;
                     00028 ;        PIC to Serial EE Connections:
                     00029 ;
                     00030 ;          PIC            Serial EE
                     00031 ;       ------------     ------------
                     00032 ;       Pin 10 (RB4) --> Chip Select
                     00033 ;       Pin 11 (RB5) --> Clock
                     00034 ;       Pin 12 (RB6) --> Data In
                     00035 ;       Pin 13 (RB7) --> Data Out
                     00036 ;                        ORG = Vcc
                     00037 ;
                     00038 ;
                     00039 ;       Program:          3WBYTER.ASM
                     00040 ;       Revision Date:
                     00041 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00042 ;
                     00043 ;*****************************************************************
                     00044 ;             Register Assignments
                     00045 ;*****************************************************************
00000003             00046 status  equ     3h      ; status register
00000005             00047 port_a  equ     5h      ; port 5 (port_a)
00000006             00048 port_b  equ     6h      ; port 6 (port b) comm lines to serial EE
0000000A             00049 eeprom  equ     0ah     ; bit buffer
0000000C             00050 addr    equ     0ch     ; address register
0000000D             00051 datai   equ     0dh     ; stored data input reg.
0000000E             00052 datao   equ     0eh     ; stored data output reg.
00000010             00053 txbuf   equ     10h     ; transmit buffer
00000011             00054 count   equ     11h     ; bits transmitted so far
```


© 1997 Microchip Technology Inc.

```
  00000012          00055 bits    equ    12h      ; bits to transmit
  00000013          00056 bytcnt  equ    13h      ; byte counter for read routine
  00000015          00057 loops   equ    15h      ; delay loop counter
  00000016          00058 loops2  equ    16h      ; delay loop counter
  00000017          00059 hbyte   equ    17h      ; high byte for input data
  00000018          00060 lbyte   equ    18h      ; low byte for input data
                    00061 ;*****************************************************
                    00062 ;             Bit Assignments
                    00063 ;*****************************************************
  00000007          00064 di      equ    7        ; eeprom input
  00000006          00065 do      equ    6        ; eeprom output
  00000007          00066 datout  equ    7        ; data out line (port_b)
  00000006          00067 datin   equ    6        ; data in line (port_b)
  00000005          00068 sclk    equ    5        ; clock line (port_b)
  00000004          00069 chpsel  equ    4        ; chip select line (port_b)
                    00070 ;
                    00071 ;*****************************************************
01FF                00072         org    01ffh
01FF 0A33           00073 begin   goto   PWRUP    ; set the reset vector
0000                00074         org    000h
0000 0A33           00075         goto   PWRUP
                    00076 ;
                    00077 ;*****************************************************
                    00078 ;       Start Bit Subroutine
                    00079 ;       this routine generates a start bit
                    00080 ;       (Chip select and DI high when clock goes high)
                    00081 ;*****************************************************
0001                00082 BSTART
0001 04C6           00083          bcf     port_b,datin   ; set datain and chipselect lines
0002 0486           00084          bcf     port_b,chpsel  ; low just to check operation
0003 04A6           00085          bcf     port_b,sclk    ; make sure clock starts low too.
0004 0000           00086          nop
                    00087 ;
0005 0586           00088          bsf     port_b,chpsel  ; set chip select line high
0006 05C6           00089          bsf     port_b,datin   ; set data in line high
0007 0000           00090          nop
0008 05A6           00091          bsf     port_b,sclk    ; set the clock line high to
                    00092                                 ; generate the start bit
0009 0000           00093          nop
000A 0000           00094          nop
000B 04A6           00095          bcf     port_b,sclk    ; set clock low again
000C 0800           00096          retlw   0
                    00097          ;
                    00098 ;*****************************************************
                    00099 ;       BITIN routine reads one bit of data from the
                    00100 ;       serial EE device and stores it in 'di'
                    00101 ;*****************************************************
000D                00102 BITIN
000D 05EA           00103          bsf     eeprom,di      ; assume input bit is high
000E 05A6           00104          bsf     port_b,sclk    ; set clock line high
000F 0000           00105          nop                    ;
0010 07E6           00106          btfss   port_b,datout  ; read the data bit
0011 04EA           00107          bcf     eeprom,di      ; input bit was low
0012 04A6           00108          bcf     port_b,sclk    ; set clock line low
                    00109                                 ;
0013 0800           00110          retlw   0              ;
                    00111
                    00112 ;*****************************************************
                    00113 ;       Receive data routine
                    00114 ;       This routine reads one byte of data from the part
                    00115 ;       into the 'datai' register.
                    00116 ;*****************************************************
0014                00117 RX
0014 006D           00118          clrf    datai          ; clear input buffer
0015 0C08           00119          movlw   .8             ; set # bits to 8
0016 0031           00120          movwf   count
```


© 1997 Microchip Technology Inc.

```
0017 0403          00121          bcf     status,0        ; make sure carry bit is low
0018 036D          00122 RXLP     rlf     datai, F        ; rotate the buffer left 1 bit
0019 090D          00123          call    BITIN           ; read 1 bit
001A 040D          00124          bcf     datai,0         ; assume the input bit was low
001B 06EA          00125          btfsc   eeprom,di       ; check the bit
001C 050D          00126          bsf     datai,0         ; set high if neccessary
001D 02F1          00127          decfsz  count, F        ; 8 bits done?
001E 0A18          00128          goto    RXLP            ; no, do another
001F 0800          00129          retlw   0
                   00130 ;
                   00131 ;*************************************************************
                   00132 ;       BITOUT routine
                   00133 ;       This routine takes one bit of data in 'do' and
                   00134 ;       transmits it to the serial EE device
                   00135 ;*************************************************************
0020               00136 BITOUT
0020 07CA          00137          btfss   eeprom,do       ; check state of data bit
0021 0A24          00138          goto    bitlow          ; low, goto bitlow
0022 05C6          00139          bsf     port_b,datin    ; high, set datain high
0023 0A25          00140          goto    clkout          ; and clock it
                   00141                                  ;
0024 04C6          00142 bitlow   bcf     port_b,datin    ; output a logic low
0025 05A6          00143 clkout   bsf     port_b,sclk     ; set clock line high
0026 0000          00144          nop
0027 04A6          00145          bcf     port_b,sclk     ; return clock line low
0028 0800          00146          retlw   0
                   00147 ;
                   00148 ;*************************************************************
                   00149 ;       Transmit Data Subroutine
                   00150 ;       This routine takes the byte of data stored in the
                   00151 ;       'datao' register and transmits it to the serial EE device.
                   00152 ;*************************************************************
0029               00153 TX
0029 0212          00154          movf    bits,w          ; set the number of bits to xmit
002A 0031          00155          movwf   count
                   00156          ;
002B               00157 TXLP
002B 04CA          00158          bcf     eeprom,do       ; assume bit 7 is low
002C 06F0          00159          btfsc   txbuf,7         ; is bit 7 clear?
002D 05CA          00160          bsf     eeprom,do       ; no, set data bit =1
002E 0920          00161          call    BITOUT          ; transmit 1 bit to serial EE
002F 0370          00162          rlf     txbuf, F        ; rotate txbuf left
0030 02F1          00163          decfsz  count, F        ; all bits done?
0031 0A2B          00164          goto    TXLP            ; no, do another bit
0032 0800          00165          retlw   0               ; yes, jump out
                   00166 ;
                   00167 ;*************************************************************
                   00168 ;            POWER-UP ROUTINE
                   00169 ;       This is the program entry point, which in this case simply
                   00170 ;       sets the port_a I/O lines and directs control to the
                   00171 ;       read routine.
                   00172 ;*************************************************************
0033               00173 PWRUP
                   00174 ;
0033 0C00          00175          movlw   b'00000000'
0034 0005          00176          tris    port_a          ; set port_a as all output
0035 0065          00177          clrf    port_a          ; all lines low
                   00178
0036 0C80          00179          movlw   b'10000000'
0037 0006          00180          tris    port_b          ; set RB7 as input, rest output;
                   00181 ;
                   00182 ;               Fall through and do the read
                   00183 ;
                   00184 ;*************************************************************
                   00185 ;       READ ROUTINE
                   00186 ;       This routine reads 8 consecutive addresses in
```


© 1997 Microchip Technology Inc.

```
                    00187 ;       random mode starting at address 0.  This is done in
                    00188 ;       x16 mode and will repeat forever.
                    00189 ;*****************************************************************
0038                00190 READ
                    00191 ;
0038 0C00           00192         movlw   .0              ; set starting address to 00
0039 002C           00193         movwf   addr            ;
003A 0C08           00194         movlw   .8              ; set number of addresses to
003B 0033           00195         movwf   bytcnt          ; read as 8
                    00196                                 ;
003C 0901           00197 rbyte   call    BSTART          ; generate the start bit
003D 0C02           00198         movlw   .2              ; set # bits to 2
003E 0032           00199         movwf   bits            ;
003F 0C80           00200         movlw   b'10000000'     ; get opcode (10b)
0040 0030           00201         movwf   txbuf           ; into output buffer
0041 0929           00202         call    TX              ; and transmit it
0042 0C08           00203         movlw   .8              ;
0043 0032           00204         movwf   bits            ; set number of bits to 8
0044 020C           00205         movf    addr,w          ; get the address
0045 0030           00206         movwf   txbuf           ; into the output buffer
0046 0929           00207         call    TX              ; and transmit it
                    00208
0047 0914           00209         call    RX              ; read the high byte
0048 020D           00210         movf    datai,w         ; move input data to w
0049 0037           00211         movwf   hbyte           ; xfer it to high byte
                    00212
004A 0914           00213         call    RX              ; read the low byte
004B 020D           00214         movf    datai,w         ; move input data to w
004C 0037           00215         movwf   hbyte           ; xfer it to low byte
                    00216
004D 0486           00217         bcf     port_b,chpsel   ; clear the chip select line
                    00218
004E 02AC           00219         incf    addr, F         ; add 1 to the address
004F 02F3           00220         decfsz  bytcnt, F       ; have all bytes been read?
0050 0A3C           00221         goto    rbyte           ; no, read another byte
0051 0A38           00222         goto    READ            ; yes, start over
                    00223     END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XX-------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:    83
Program Memory Words Free:   429

Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

7

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: 3WBYTEW.ASM

```
MPASM 01.40 Released          3WBYTEW.ASM   1-16-1997  16:39:06         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54
                     00002 ;*****************************************************************
                     00003 ;       3-Wire Byte Write Program (106 bytes)
                     00004 ;
                     00005 ;       This program demonstrates how to interface a
                     00006 ;       Microchip PIC 16C54 to a 93LC56 or 93LC66 Serial EE
                     00007 ;       device. This program will execute the erase/write enable
                     00008 ;       command, write to 8 consecutive addresses, and then
                     00009 ;       execute the erase/write disable command.  This
                     00010 ;       sequence will repeat forever.
                     00011 ;
                     00012 ;       After each byte is written, time must be given to the
                     00013 ;       device for it to complete the write cycle before
                     00014 ;       the next command can be sent.  The easiest solution
                     00015 ;       is to consult the data book for the maximum write
                     00016 ;       cycle time and just wait that long before the next
                     00017 ;       command is sent.  This program demonstrates that
                     00018 ;       solution.
                     00019 ;
                     00020 ;       Another, more efficient method of determining when the
                     00021 ;       write cycle is complete is called 'data polling.'  This
                     00022 ;       method is demonstrated in the program "3wdpoll."
                     00023 ;
                     00024 ;       This program communicates to the serial EE in the
                     00025 ;       x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
                     00026 ;       ON THE DEVICE TO Vcc.
                     00027 ;
                     00028 ;       Timing is based on using the PIC in 'XT' mode
                     00029 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00030 ;       will be approximately 40 Khz for this setup.
                     00031 ;
                     00032 ;        PIC to Serial EE Connections:
                     00033 ;
                     00034 ;          PIC            Serial EE
                     00035 ;       ------------     ------------
                     00036 ;       Pin 10 (RB4) --> Chip Select
                     00037 ;       Pin 11 (RB5) --> Clock
                     00038 ;       Pin 12 (RB6) --> Data In
                     00039 ;       Pin 13 (RB7) --> Data Out
                     00040 ;                        ORG=Vcc
                     00041 ;
                     00042 ;
                     00043 ;       Program:          3WBYTEW.ASM
                     00044 ;       Revision Date:
                     00045 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00046 ;
                     00047 ;*****************************************************************
                     00048 ;             Register Assignments
                     00049 ;*****************************************************************
00000005             00050 port_a  equ     5h      ; port 5 (port_a)
00000006             00051 port_b  equ     6h      ; port 6 (port b) comm lines to serial EE
0000000A             00052 eeprom  equ    0ah      ; bit buffer
0000000C             00053 addr    equ    0ch      ; address register
0000000D             00054 datai   equ    0dh      ; stored data input reg.
```


© 1997 Microchip Technology Inc.

```
  0000000E          00055 datao   equ    0eh     ; stored data output reg.
  00000010          00056 txbuf   equ    10h     ; transmit buffer
  00000011          00057 count   equ    11h     ; bits transmitted so far
  00000012          00058 bits    equ    12h     ; bits to transmit
  00000013          00059 bytcnt  equ    13h     ; byte counter for write routine
  00000015          00060 loops   equ    15h     ; delay loop counter
  00000016          00061 loops2  equ    16h     ; delay loop counter        ;
                    00062 ;*************************************************************
                    00063 ;             Bit Assignments
                    00064 ;*************************************************************
  00000007          00065 di      equ    7       ; eeprom input
  00000006          00066 do      equ    6       ; eeprom output
  00000007          00067 datout  equ    7       ; data out line (port_b)
  00000006          00068 datin   equ    6       ; data in line (port_b)
  00000005          00069 sclk    equ    5       ; clock line (port_b)
  00000004          00070 chpsel  equ    4       ; chip select line (port_b)
                    00071 ;
                    00072 ;*************************************************************
01FF                00073         org    01ffh
01FF 0A2F           00074 begin   goto   PWRUP   ; set the reset vector
0000                00075         org    000h
0000 0A2F           00076         goto   PWRUP
                    00077 ;
                    00078 ;*************************************************************
                    00079 ;       DELAY ROUTINE
                    00080 ;       This routine takes the value in 'loops'
                    00081 ;       and multiplies it times 1 millisecond to
                    00082 ;       determine delay time.
                    00083 ;*************************************************************
0001                00084 WAIT
                    00085 ;
0001 0C6E           00086 top      movlw   .110           ; timing adjustment variable
0002 0036           00087          movwf   loops2
0003 0000           00088 top2     nop                    ; sit and wait
0004 0000           00089          nop
0005 0000           00090          nop
0006 0000           00091          nop
0007 0000           00092          nop
0008 0000           00093          nop
0009 02F6           00094          decfsz  loops2, F      ; inner loops complete?
000A 0A03           00095          goto    top2           ; no, go again
                    00096                                 ;
000B 02F5           00097          decfsz  loops, F       ; outer loops complete?
000C 0A01           00098          goto    top            ; no, go again
000D 0800           00099          retlw   0              ; yes, return from sub
                    00100 ;
                    00101 ;*************************************************************
                    00102 ;       Start Bit Subroutine
                    00103 ;       this routine generates a start bit
                    00104 ;       (Chip select and DI high when clock goes high)
                    00105 ;*************************************************************;
000E                00106 BSTART
000E 0C8F           00107          movlw   b'10001111'
000F 0006           00108          tris    port_b         ; set port b for output
                    00109                                 ; except for the data out line
0010 04C6           00110          bcf     port_b,datin   ; set datain and chipselect lines
0011 0486           00111          bcf     port_b,chpsel  ; low just to check operation
0012 04A6           00112          bcf     port_b,sclk    ; make sure clock starts low too.
0013 0000           00113          nop
                    00114 ;
0014 0586           00115          bsf     port_b,chpsel  ; set chip select line high
0015 05C6           00116          bsf     port_b,datin   ; set data in line high
0016 0000           00117          nop
0017 05A6           00118          bsf     port_b,sclk    ; set the clock line high to
                    00119                                 ; generate the start bit
0018 0000           00120          nop
```


© 1997 Microchip Technology Inc.

```
0019 0000           00121          nop
001A 04A6           00122          bcf     port_b,sclk     ; set clock low again
001B 0800           00123          retlw   0
                    00124          ;
                    00125 ;*****************************************************************
                    00126 ;        BITOUT routine
                    00127 ;        This routine takes one bit of data in 'do' and
                    00128 ;        transmits it to the serial EE device
                    00129 ;*****************************************************************
001C                00130 BITOUT
001C 07CA           00131          btfss   eeprom,do       ; check state of data bit
001D 0A20           00132          goto    bitlow          ; low, goto bitlow
001E 05C6           00133          bsf     port_b,datin    ; high, set datain high
001F 0A21           00134          goto    clkout          ; and clock it
                    00135                                  ;
0020 04C6           00136 bitlow   bcf     port_b,datin    ; output a logic low
0021 05A6           00137 clkout   bsf     port_b,sclk     ; set clock line high
0022 0000           00138          nop
0023 04A6           00139          bcf     port_b,sclk     ; return clock line low
0024 0800           00140          retlw   0
                    00141 ;
                    00142 ;*****************************************************************
                    00143 ;        Transmit Data Subroutine
                    00144 ;        This routine takes the byte of data stored in the
                    00145 ;        'datao' register and transmits it to the serial EE device.
                    00146 ;*****************************************************************
0025                00147 TX
0025 0212           00148          movf    bits,w          ; set the number of bits to xmit
0026 0031           00149          movwf   count
                    00150          ;
0027                00151 TXLP
0027 04CA           00152          bcf     eeprom,do       ; assume bit 7 is low
0028 06F0           00153          btfsc   txbuf,7         ; is bit 7 clear?
0029 05CA           00154          bsf     eeprom,do       ; no, set data bit =1
002A 091C           00155          call    BITOUT          ; transmit 1 bit to serial EE
002B 0370           00156          rlf     txbuf, F        ; rotate txbuf left
002C 02F1           00157          decfsz  count, F        ; all bits done?
002D 0A27           00158          goto    TXLP            ; no, do another bit
002E 0800           00159          retlw   0               ; yes, jump out
00160 ;
                    00161 ;*****************************************************************
                    00162 ;             POWER-UP ROUTINE
                    00163 ;        This is the program entry point, which in this case simply
                    00164 ;        sets the port_a I/O lines and directs control to the
                    00165 ;        erase/write enable routine.
                    00166 ;*****************************************************************
002F                00167 PWRUP
                    00168 ;
002F 0C00           00169          movlw   b'00000000'
0030 0005           00170          tris    port_a          ; set port_a as all output
0031 0065           00171          clrf    port_a          ; all lines low
                    00172
0032 0C80           00173          movlw   b'10000000'
0033 0006           00174          tris    port_b          ; set RB7 as input, rest output;
                    00175 ;
                    00176 ;        Fall through and do erase/write enable
                    00177 ;
                    00178 ;*****************************************************************
                    00179 ;          EWEN (Erase/Write ENable Routine)
                    00180 ;            this routine enables the dut for erasing and
                    00181 ;            writing. This must be done prior to any erase,write
                    00182 ;            eral,wral instructions.
                    00183 ;*****************************************************************
0034                00184 EWEN
                    00185 ;
0034 090E           00186          call    BSTART          ; generate a start bit
```


© 1997 Microchip Technology Inc.

```
                    00187                                ;
0035 0C02           00188           movlw   .2           ; set # bits to 2
0036 0032           00189           movwf   bits         ;
0037 0C00           00190           movlw   b'00000000'  ; get the opcode (00b)
0038 0030           00191           movwf   txbuf        ; into the output buffer
0039 0925           00192           call    TX           ; and transmit it
003A 0C08           00193           movlw   .8           ; set # bits to 8
003B 0032           00194           movwf   bits         ;
003C 0CC0           00195           movlw   b'11000000'  ; get opcode and address
                    00196                                ;     (11XXXXXX)
003D 0030           00197           movwf   txbuf        ; into output buffer
003E 0925           00198           call    TX           ; and transmit it
003F 0486           00199           bcf     port_b,chpsel ; set chip select line low
0040 0000           00200           nop
                    00201 ;
                    00202 ;         Now continue on to the write command
                    00203 ;
                    00204 ;*****************************************************************
                    00205 ;         Byte Write Routine
                    00206 ;         This routine writes an AA55h pattern into
                    00207 ;         8 consecutive addresses starting at address 00.
                    00208 ;         A delay of about 10ms is given after each byte
                    00209 ;         for the write cycle to complete.
                    00210 ;         The write is done in the x16 mode: the user must
                    00211 ;         have the ORG pin tied to Vcc on the device
                    00212 ;*****************************************************************
0041                00213 WRITE
                    00214 ;
0041 0C00           00215           movlw   .0           ; set starting address to 00
0042 002C           00216           movwf   addr         ;
0043 0C08           00217           movlw   .8           ; set number of bytes to write as 8
0044 0033           00218           movwf   bytcnt       ;
                    00219                                ;
0045 090E           00220 topwr     call    BSTART       ; generate the start bit
                    00221                                ;
0046 0C02           00222           movlw   .2           ; set # bits to 2 for the opcode
0047 0032           00223           movwf   bits         ;
0048 0C40           00224           movlw   b'01000000'  ; get opcode (01b)
0049 0030           00225           movwf   txbuf        ; into the transmit buffer
004A 0925           00226           call    TX           ; and send it
                    00227                                ;
004B 0C08           00228           movlw   .8           ; set # of bits to 8 for the
004C 0032           00229           movwf   bits         ; address
004D 020C           00230           movf    addr,w       ; get address counter
004E 0030           00231           movwf   txbuf        ; into output buffer
004F 0925           00232           call    TX           ; and send it
0050 0CAA           00233           movlw   b'10101010'  ; get upper byte of data (AAh)
0051 0030           00234           movwf   txbuf        ; into the transmit buffer
0052 0925           00235           call    TX           ; and send it
0053 0C55           00236           movlw   b'01010101'  ; get lower byte of data (55h)
0054 0030           00237           movwf   txbuf        ; into transmit buffer
0055 0925           00238           call    TX           ; and send it
                    00239                                ;
0056 0486           00240           bcf     port_b,chpsel ; clear the chip select line
                    00241                                ; to initiate write cycle
                    00242                                ;
0057 0C0A           00243           movlw   .10          ;
0058 0035           00244           movwf   loops        ; set delay time to 10mS
0059 0901           00245           call    WAIT         ; and wait
                    00246                                ;
005A 02AC           00247           incf    addr, F      ; increment address counter
005B 02F3           00248           decfsz  bytcnt, F    ; all bytes written?
005C 0A45           00249           goto    topwr        ; no, do another
                    00250                                ; yes, go on
                    00251 ;
                    00252 ;     Now continue on to the erase/write disable command
```


© 1997 Microchip Technology Inc.

```
                    00253 ;
                    00254 ;*****************************************************************
                    00255 ;       EWDS (Erase/Write Disable Routine)
                    00256 ;       This routine executes the erase/write disable command
                    00257 ;       which prevents the contents of the array from being
                    00258 ;       written to.
                    00259 ;
                    00260 ;*****************************************************************
005D                00261 EWDS
                    00262 ;
005D 090E           00263         call    BSTART          ; generate a start bit
                    00264 ;
005E 0C02           00265         movlw   .2              ; set # bits to 2
005F 0032           00266         movwf   bits            ;
0060 0C00           00267         movlw   b'00000000'     ; get the opcode (00b)
0061 0030           00268         movwf   txbuf           ; into the output buffer
0062 0925           00269         call    TX              ; and transmit it
0063 0C08           00270         movlw   .8              ; set # bits to 8
0064 0032           00271         movwf   bits            ;
0065 0C00           00272         movlw   b'00000000'     ; get opcode and address
                    00273                                 ;     (00XXXXXX)
0066 0030           00274         movwf   txbuf           ; into output buffer
0067 0925           00275         call    TX              ; and transmit it
0068 0486           00276         bcf     port_b,chpsel   ; set chip select line low
0069 0000           00277         nop
006A 0A34           00278         goto    EWEN            ; start all over
                    00279 ;
                    00280     END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXX----- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   108
Program Memory Words Free:   404
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C: 3WDPOLL.ASM

```
MPASM 01.40 Released          3WDPOLL.ASM   1-16-1997  16:39:36         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54
                     00002 ;*****************************************************************
                     00003 ;       3-Wire Byte Write with Data Poll Program (107 bytes)
                     00004 ;
                     00005 ;       This program demonstrates how to interface a
                     00006 ;       Microchip PIC 16C54 to a 93LC56 or 93LC66 Serial EE
                     00007 ;       device. This program will execute the erase/write enable
                     00008 ;       command, write to 8 consecutive addresses, and then
                     00009 ;       use the 'data polling' method to determine when the
                     00010 ;       write cycle is complete.  The program then executes
                     00011 ;       the erase/write disable command.  This sequence will
                     00012 ;       repeat forever.
                     00013 ;
                     00014 ;       When writing to a 3-wire serial EE device, the
                     00015 ;       internally timed write cycle will begin after
                     00016 ;       the opcode, address and data are sent to the
                     00017 ;       device.  There are two ways to make sure that the
                     00018 ;       cycle is complete before you send it another command.
                     00019 ;       The simplest method is to simply wait for the maximum
                     00020 ;       write cycle time, which can be obtained from the data book.
                     00021 ;       A more efficient method is "Data polling."  This is
                     00022 ;       done by toggling the chip select line low and then
                     00023 ;       back high after the opcode, address and data are sent.
                     00024 ;       The chip select line must be low for at least 250ns
                     00025 ;       before bringing it high again.  The device will pull
                     00026 ;       the data out line low at that point, and set it high
                     00027 ;       when the write cycle is complete.  The user can then check
                     00028 ;       or "poll" the dataout line until it goes high before
                     00029 ;       sending the next command.
                     00030 ;
                     00031 ;       As an option, the user can connect a LED to pin 18
                     00032 ;       on the PIC (use about a 1K resistor in series) as a
                     00033 ;       timeout indicator.  This LED will come on if the
                     00034 ;       data polling is unsuccessful and the device being
                     00035 ;       programmed does not respond.  This can be tested by
                     00036 ;       simply running the program with no part in the socket.
                     00037 ;
                     00038 ;       This program communicates to the serial EE in the
                     00039 ;       x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
                     00040 ;       ON THE DEVICE TO Vcc.
                     00041 ;
                     00042 ;       Timing is based on using the PIC in 'XT' mode
                     00043 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00044 ;       will be approximately 50 Khz for this setup.
                     00045 ;
                     00046 ;        PIC to Serial EE Connections:
                     00047 ;
                     00048 ;           PIC             Serial EE
                     00049 ;       ------------     ------------
                     00050 ;       Pin 10 (RB4) --> Chip Select
                     00051 ;       Pin 11 (RB5) --> Clock
                     00052 ;       Pin 12 (RB6) --> Data In
                     00053 ;       Pin 13 (RB7) --> Data Out
                     00054 ;                        ORG=Vcc
```


© 1997 Microchip Technology Inc.

```
                    00055 ;
                    00056 ;
                    00057 ;       Program:          3WDPOLL.ASM
                    00058 ;       Revision Date:
                    00059 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                    00060 ;
                    00061 ;*****************************************************************
                    00062 ;             Register Assignments
                    00063 ;*****************************************************************
  00000005          00064 port_a  equ     5h      ; port 5 (port_a)
  00000006          00065 port_b  equ     6h      ; port 6 (port b) used comm lines to serial EE
  0000000A          00066 eeprom  equ    0ah      ; bit buffer
  0000000C          00067 addr    equ    0ch      ; address register
  0000000D          00068 datai   equ    0dh      ; stored data input reg.
  0000000E          00069 datao   equ    0eh      ; stored data output reg.
  00000010          00070 txbuf   equ    10h      ; transmit buffer
  00000011          00071 count   equ    11h      ; bits transmitted so far
  00000012          00072 bits    equ    12h      ; bits to transmit
  00000013          00073 bytcnt  equ    13h      ; byte counter for write routine
  00000015          00074 loops   equ    15h      ; delay loop counter
  00000016          00075 loops2  equ    16h      ; delay loop counter
                    00076 ;*****************************************************************
                    00077 ;             Bit Assignments
                    00078 ;*****************************************************************
  00000007          00079 di      equ    7        ; eeprom input
  00000006          00080 do      equ    6        ; eeprom output
  00000007          00081 datout  equ    7        ; data out line (port_b)
  00000006          00082 datin   equ    6        ; data in line (port_b)
  00000005          00083 sclk    equ    5        ; clock line (port_b)
  00000004          00084 chpsel  equ    4        ; chip select line (port_b)
  00000001          00085 timeout equ    1        ; write cycle timeout warning (port_a)
                    00086 ;
                    00087 ;*****************************************************************
01FF                00088         org    01ffh
01FF 0A26           00089 begin   goto   PWRUP    ; set the reset vector
0000                00090         org    000h
0000 0A26           00091         goto   PWRUP
                    00092 ;
                    00093 ;*****************************************************************
                    00094 ;       DATA POLL DELAY ROUTINE
                    00095 ;       This routine delays 100 us is
                    00096 ;       used for delay between polls
                    00097 ;*****************************************************************
0001                00098 dpdelay
                    00099 ;
0001 0C19           00100          movlw   .25            ; timing adjustment variable
0002 0036           00101          movwf   loops2
0003 0000           00102 top      nop
0004 02F6           00103          decfsz  loops2, F      ; loops complete?
0005 0A03           00104          goto    top            ; no, go again
                    00105                                 ;
0006 0800           00106          retlw   0              ; yes, return from sub
                    00107 ;
                    00108 ;*****************************************************************
                    00109 ;       Start Bit Subroutine
                    00110 ;       this routine generates a start bit
                    00111 ;       (Chip select and DI high when clock goes high)
                    00112 ;*****************************************************************;
0007                00113 BSTART
0007 04C6           00114          bcf     port_b,datin   ; set datain and chipselect lines
0008 0486           00115          bcf     port_b,chpsel  ; low just to check operation
0009 04A6           00116          bcf     port_b,sclk    ; make sure clock starts low too.
000A 0000           00117          nop
                    00118 ;
000B 0586           00119          bsf     port_b,chpsel  ; set chip select line high
000C 05C6           00120          bsf     port_b,datin   ; set data in line high
```


© 1997 Microchip Technology Inc.

```
000D 0000           00121           nop
000E 05A6           00122           bsf     port_b,sclk     ; set the clock line high to
                    00123                                   ; generate the start bit
000F 0000           00124           nop
0010 0000           00125           nop
0011 04A6           00126           bcf     port_b,sclk     ; set clock low again
0012 0800           00127           retlw   0
                    00128           ;
                    00129 ;*****************************************************************
                    00130 ;       BITOUT routine
                    00131 ;       This routine takes one bit of data in 'do' and
                    00132 ;       transmits it to the serial EE device
                    00133 ;*****************************************************************
0013                00134 BITOUT
0013 07CA           00135           btfss   eeprom,do       ; check state of data bit
0014 0A17           00136           goto    bitlow          ; low, goto bitlow
0015 05C6           00137           bsf     port_b,datin    ; high, set datain high
0016 0A18           00138           goto    clkout          ; and clock it
                    00139                                   ;
0017 04C6           00140 bitlow    bcf     port_b,datin    ; output a logic low
0018 05A6           00141 clkout    bsf     port_b,sclk     ; set clock line high
0019 0000           00142           nop
001A 04A6           00143           bcf     port_b,sclk     ; return clock line low
001B 0800           00144           retlw   0
                    00145 ;
                    00146 ;*****************************************************************
                    00147 ;       Transmit Data Subroutine
                    00148 ;       This routine takes the byte of data stored in the
                    00149 ;       'datao' register and transmits it to the serial EE device.
                    00150 ;*****************************************************************
001C                00151 TX
001C 0212           00152           movf    bits,w          ; set the number of bits to xmit
001D 0031           00153           movwf   count
                    00154           ;
001E                00155 TXLP
001E 04CA           00156           bcf     eeprom,do       ; assume bit 7 is low
001F 06F0           00157           btfsc   txbuf,7         ; is bit 7 clear?
0020 05CA           00158           bsf     eeprom,do       ; no, set data bit =1
0021 0913           00159           call    BITOUT          ; transmit 1 bit to serial EE
0022 0370           00160           rlf     txbuf, F        ; rotate txbuf left
0023 02F1           00161           decfsz  count, F        ; all bits done?
0024 0A1E           00162           goto    TXLP            ; no, do another bit
0025 0800           00163           retlw   0               ; yes, jump out
                    00164 ;
                    00165 ;*****************************************************************
                    00166 ;           POWER-UP ROUTINE
                    00167 ;       This is the program entry point, which in this case simply
                    00168 ;       sets the port_a I/O lines and directs control to the
                    00169 ;       erase/write enable routine.
                    00170 ;*****************************************************************
0026                00171 PWRUP
                    00172 ;
0026 0C00           00173           movlw   b'00000000'
0027 0005           00174           tris    port_a          ; set port_a as all output
0028 0065           00175           clrf    port_a          ; all lines low
                    00176
0029 0C80           00177           movlw   b'10000000'
002A 0006           00178           tris    port_b          ; set RB7 as input, rest output;
                    00179 ;
                    00180 ;       Fall through and do erase/write enable
                    00181 ;
                    00182 ;*****************************************************************
                    00183 ;         EWEN (Erase/Write ENable Routine)
                    00184 ;           this routine enables the dut for erasing and
                    00185 ;           writing. This must be done prior to any erase,write
                    00186 ;           eral,wral instructions.
```


© 1997 Microchip Technology Inc.

```
                00187 ;*****************************************************************
002B            00188 EWEN
                00189 ;
002B 0907       00190         call    BSTART          ; generate a start bit
                00191                                 ;
002C 0C02       00192         movlw   .2              ; set # bits to 2
002D 0032       00193         movwf   bits            ;
002E 0C00       00194         movlw   b'00000000'     ; get the opcode (00b)
002F 0030       00195         movwf   txbuf           ; into the output buffer
0030 091C       00196         call    TX              ; and transmit it
0031 0C08       00197         movlw   .8              ; set # bits to 8
0032 0032       00198         movwf   bits            ;
0033 0CC0       00199         movlw   b'11000000'     ; get opcode and address
                00200                                 ;     (11XXXXXX)
0034 0030       00201         movwf   txbuf           ; into output buffer
0035 091C       00202         call    TX              ; and transmit it
0036 0486       00203         bcf     port_b,chpsel   ; set chip select line low
0037 0000       00204         nop
                00205 ;
                00206 ;       Now continue on to the write command
                00207 ;
                00208 ;*****************************************************************
                00209 ;       WRITE
                00210 ;       This routine writes a AA55h pattern into
                00211 ;       8 consecutive addresses starting at address 00.
                00212 ;       It will then 'poll' the data out line to determine
                00213 ;       when the write cycle is complete.  If the cycle has
                00214 ;       not completed within 20 ms, then it will continue
                00215 ;       anyway and turn the timeout fail LED on.
                00216 ;       The write is done in the x16 mode: the user must
                00217 ;       have the ORG pin tied to Vcc on the device. This
                00218 ;       program will repeat forever.
                00219 ;*****************************************************************
0038            00220 WRITE
                00221 ;
0038 0C00       00222         movlw   .0              ; set starting address to 00
0039 002C       00223         movwf   addr            ;
003A 0C08       00224         movlw   .8              ; set number of bytes to write as 8
003B 0033       00225         movwf   bytcnt          ;
                00226                                 ;
003C 0907       00227 topwr   call    BSTART          ; generate the start bit
                00228                                 ;
003D 0C02       00229         movlw   .2              ; set # bits to 2 for the opcode
003E 0032       00230         movwf   bits            ;
003F 0C40       00231         movlw   b'01000000'     ; get opcode (01b)
0040 0030       00232         movwf   txbuf           ; into the transmit buffer
0041 091C       00233         call    TX              ; and send it
                00234                                 ;
0042 0C08       00235         movlw   .8              ; set # of bits to 8 for the
0043 0032       00236         movwf   bits            ; address
0044 020C       00237         movf    addr,w          ; get address counter
0045 0030       00238         movwf   txbuf           ; into output buffer
0046 091C       00239         call    TX              ; and send it
0047 0CAA       00240         movlw   b'10101010'     ; get first half of data (AAh)
0048 0030       00241         movwf   txbuf           ; into the transmit buffer
0049 091C       00242         call    TX              ; and send it
004A 0C55       00243         movlw   b'01010101'     ; get second half of data (55h)
004B 0030       00244         movwf   txbuf           ; into transmit buffer
004C 091C       00245         call    TX              ; and send it
                00246                                 ;
004D 0486       00247         bcf     port_b,chpsel   ; clear the chip select line
                00248                                 ; to initiate write cycle
004E 0000       00249         nop
004F 0586       00250         bsf     port_b,chpsel   ; set the chip sel line high
                00251                                 ; to begin data polling
0050 0CC8       00252         movlw   .200            ;
```


© 1997 Microchip Technology Inc.

```
0051 0035           00253          movwf   loops          ; poll 100 times before timeout(about 20ms)
0052 0901           00254 polltop  call    dpdelay        ; wait 100us
0053 06E6           00255          btfsc   port_b,datout  ; is the data out line high?
0054 0A59           00256          goto    okpoll         ; yes-cycle complete: do another
0055 02F5           00257          decfsz  loops, F       ; has it timed out?
0056 0A52           00258          goto    polltop        ; no, check again
0057 0525           00259          bsf     port_a,timeout; yes, set timeout LED and go on
0058 0A5A           00260          goto    badpoll                      ;
                    00261
0059 0425           00262 okpoll   bcf     port_a,timeout; clear the timeout LED
005A 0486           00263 badpoll  bcf     port_b,chpsel ; chip sel line back low
005B 02AC           00264          incf    addr, F       ; increment address counter
005C 02F3           00265          decfsz  bytcnt, F     ; all bytes written?
005D 0A3C           00266          goto    topwr         ; no, do another
                    00267                                ; yes, go on
                    00268 ;
                    00269 ;     Now continue on to the erase/write disable command
                    00270 ;
                    00271 ;*****************************************************************
                    00272 ;          EWDS (Erase/Write Disable Routine)
                    00273 ;       This routine executes the erase/write disable command
                    00274 ;       which prevents the contents of the array from being
                    00275 ;       written to.
                    00276 ;
                    00277 ;*****************************************************************
005E                00278 EWDS
                    00279 ;
005E 0907           00280          call    BSTART         ; generate a start bit
                    00281 ;
005F 0C02           00282          movlw   .2             ; set # bits to 2
0060 0032           00283          movwf   bits           ;
0061 0C00           00284          movlw   b'00000000'    ; get the opcode (00b)
0062 0030           00285          movwf   txbuf          ; into the output buffer
0063 091C           00286          call    TX             ; and transmit it
0064 0C08           00287          movlw   .8             ; set # bits to 8
0065 0032           00288          movwf   bits           ;
0066 0C00           00289          movlw   b'00000000'    ; get opcode and address
                    00290                                 ;     (00XXXXXX)
0067 0030           00291          movwf   txbuf          ; into output buffer
0068 091C           00292          call    TX             ; and transmit it
0069 0486           00293          bcf     port_b,chpsel  ; set chip select line low
006A 0000           00294          nop
006B 0A2B           00295          goto    EWEN           ; start all over
                    00296 ;
                    00297 ;
                    00298
                    00299      END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXX---- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   109
Program Memory Words Free:   403
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```

7

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX D: 3WSEQR.ASM

```
MPASM 01.40 Released          3WSEQR.ASM   1-16-1997  16:39:58         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54
                     00002 ;*****************************************************************
                     00003 ;       3-Wire Sequential Read Program (93 bytes)
                     00004 ;
                     00005 ;       This program demonstrates how to interface a
                     00006 ;       Microchip PIC 16C54 to a 93LC56 or 93LC66 Serial EE
                     00007 ;       device.  This program will read 8 consecutive addresses
                     00008 ;       in the 'sequential read' mode.  This means that the opcode
                     00009 ;       and address are sent for the first address only.  After
                     00010 ;       the first address is read, the Chip Select line is left
                     00011 ;       high and clocks for the remaining 7 bytes are sent to the
                     00012 ;       device. The device will automatically increment the address
                     00013 ;       pointer in this mode, allowing the user to read as many
                     00014 ;       consecutive addresses as needed. This program will repeat
                     00015 ;       forever.
                     00016 ;;
                     00017 ;       This program communicates to the serial EE in the
                     00018 ;       x16 mode, and ASSUMES THE USER HAS SET THE ORG PIN
                     00019 ;       ON THE DEVICE TO Vcc.
                     00020 ;
                     00021 ;       Timing is based on using the PIC in 'XT' mode
                     00022 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00023 ;       will be approximately 50 Khz for this setup.
                     00024 ;
                     00025 ;        PIC to Serial EE Connections:
                     00026 ;
                     00027 ;          PIC             Serial EE
                     00028 ;       ------------     ------------
                     00029 ;       Pin 10 (RB4) --> Chip Select
                     00030 ;       Pin 11 (RB5) --> Clock
                     00031 ;       Pin 12 (RB6) --> Data In
                     00032 ;       Pin 13 (RB7) --> Data Out
                     00033 ;                        ORG = Vcc
                     00034 ;
                     00035 ;
                     00036 ;       Program:          3WSEQR.ASM
                     00037 ;       Revision Date:
                     00038 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00039 ;
                     00040 ;*****************************************************************
                     00041 ;             Register Assignments
                     00042 ;*****************************************************************
00000003             00043 status  equ     3h     ; status register
00000005             00044 port_a  equ     5h     ; port 5 (port_a)
00000006             00045 port_b  equ     6h     ; port 6 (port b) used comm lines to serial EE
0000000A             00046 eeprom  equ    0ah     ; bit buffer
0000000C             00047 addr    equ    0ch     ; address register
0000000D             00048 datai   equ    0dh     ; stored data input reg.
0000000E             00049 datao   equ    0eh     ; stored data output reg.
00000010             00050 txbuf   equ    10h     ; transmit buffer
00000011             00051 count   equ    11h     ; bits transmitted so far
00000012             00052 bits    equ    12h     ; bits to transmit
00000013             00053 bytcnt  equ    13h     ; byte counter for read routine
00000015             00054 loops   equ    15h     ; delay loop counter
```


© 1997 Microchip Technology Inc.

```
00000016          00055 loops2  equ    16h     ; delay loop counter
00000017          00056 hbyte   equ    17h     ; high byte for input data
00000018          00057 lbyte   equ    18h     ; low byte for input data
                  00058 ;*************************************************************
                  00059 ;            Bit Assignments
                  00060 ;*************************************************************
00000007          00061 di      equ    7       ; eeprom input
00000006          00062 do      equ    6       ; eeprom output
00000007          00063 datout  equ    7       ; data out line (port_b)
00000006          00064 datin   equ    6       ; data in line (port_b)
00000005          00065 sclk    equ    5       ; clock line (port_b)
00000004          00066 chpsel  equ    4       ; chip select line (port_b)
                  00067 ;
                  00068 ;*************************************************************
01FF              00069         org    01ffh
01FF 0A40         00070 begin   goto   PWRUP   ; set the reset vector
0000              00071         org    000h
0000 0A40         00072         goto   PWRUP
                  00073 ;
                  00074 ;*************************************************************
                  00075 ;       DELAY ROUTINE
                  00076 ;       This routine takes the value in 'loops'
                  00077 ;       and multiplies it times 1 millisecond to
                  00078 ;       determine delay time.
                  00079 ;*************************************************************
0001              00080 WAIT
                  00081 ;
0001 0C6E         00082 top      movlw   .110           ; timing adjustment variable
0002 0036         00083          movwf   loops2
0003 0000         00084 top2     nop                    ; sit and wait
0004 0000         00085          nop
0005 0000         00086          nop
0006 0000         00087          nop
0007 0000         00088          nop
0008 0000         00089          nop
0009 02F6         00090          decfsz  loops2, F      ; inner loops complete?
000A 0A03         00091          goto    top2           ; no, go again
                  00092                                 ;
000B 02F5         00093          decfsz  loops, F       ; outer loops complete?
000C 0A01         00094          goto    top            ; no, go again
000D 0800         00095          retlw   0              ; yes, return from sub
                  00096 ;
                  00097 ;*************************************************************
                  00098 ;       Start Bit Subroutine
                  00099 ;       this routine generates a start bit
                  00100 ;       (Chip select and DI high when clock goes high)
                  00101 ;*************************************************************
000E              00102 BSTART
000E 04C6         00103          bcf     port_b,datin   ; set datain and chipselect lines
000F 0486         00104          bcf     port_b,chpsel  ; low just to check operation
0010 04A6         00105          bcf     port_b,sclk    ; make sure clock starts low too.
0011 0000         00106          nop
                  00107 ;
0012 0586         00108          bsf     port_b,chpsel  ; set chip select line high
0013 05C6         00109          bsf     port_b,datin   ; set data in line high
0014 0000         00110          nop
0015 05A6         00111          bsf     port_b,sclk    ; set the clock line high to
                  00112                                 ; generate the start bit
0016 0000         00113          nop
0017 0000         00114          nop
0018 04A6         00115          bcf     port_b,sclk    ; set clock low again
0019 0800         00116          retlw   0
                  00117          ;
                  00118 ;*************************************************************
                  00119 ;       BITIN routine reads one bit of data from the
                  00120 ;       serial EE device and stores it in 'di'
```


© 1997 Microchip Technology Inc.

```
                    00121 ;*******************************************************
001A                00122 BITIN
001A 05EA           00123         bsf     eeprom,di       ; assume input bit is high
001B 05A6           00124         bsf     port_b,sclk     ; set clock line high
001C 0000           00125         nop                     ;
001D 07E6           00126         btfss   port_b,datout   ; read the data bit
001E 04EA           00127         bcf     eeprom,di       ; input bit was low
001F 04A6           00128         bcf     port_b,sclk     ; set clock line low
                    00129                                 ;
0020 0800           00130         retlw   0               ;
                    00131
                    00132 ;*************************************************************
                    00133 ;       Receive data routine
                    00134 ;       This routine reads one byte of data from the part
                    00135 ;       into the 'datai' register.
                    00136 ;*************************************************************
0021                00137 RX
0021 006D           00138         clrf    datai           ; clear input buffer
0022 0C08           00139         movlw   .8              ; set # bits to 8
0023 0031           00140         movwf   count
0024 0403           00141         bcf     status,0        ; make sure carry bit is low
0025 036D           00142 RXLP    rlf     datai, F        ; rotate the buffer left 1 bit
0026 091A           00143         call    BITIN           ; read 1 bit
0027 040D           00144         bcf     datai,0         ; assume the input bit was low
0028 06EA           00145         btfsc   eeprom,di       ; check the bit
0029 050D           00146         bsf     datai,0         ; set high if neccessary
002A 02F1           00147         decfsz  count, F        ; 8 bits done?
002B 0A25           00148         goto    RXLP            ; no, do another
002C 0800           00149         retlw   0
                    00150 ;
                    00151 ;*******************************************************
                    00152 ;       BITOUT routine
                    00153 ;       This routine takes one bit of data in 'do' and
                    00154 ;       transmits it to the serial EE device
                    00155 ;*******************************************************
002D                00156 BITOUT
002D 07CA           00157         btfss   eeprom,do       ; check state of data bit
002E 0A31           00158         goto    bitlow          ; low, goto bitlow
002F 05C6           00159         bsf     port_b,datin    ; high, set datain high
0030 0A32           00160         goto    clkout          ; and clock it
                    00161                                 ;
0031 04C6           00162 bitlow  bcf     port_b,datin    ; output a logic low
0032 05A6           00163 clkout  bsf     port_b,sclk     ; set clock line high
0033 0000           00164         nop
0034 04A6           00165         bcf     port_b,sclk     ; return clock line low
0035 0800           00166         retlw   0
                    00167 ;
                    00168 ;*************************************************************
                    00169 ;       Transmit Data Subroutine
                    00170 ;       This routine takes the byte of data stored in the
                    00171 ;       'datao' register and transmits it to the serial EE device.
                    00172 ;*************************************************************
0036                00173 TX
0036 0212           00174         movf    bits,w          ; set the number of bits to xmit
0037 0031           00175         movwf   count
                    00176         ;
0038                00177 TXLP
0038 04CA           00178         bcf     eeprom,do       ; assume bit 7 is low
0039 06F0           00179         btfsc   txbuf,7         ; is bit 7 clear?
003A 05CA           00180         bsf     eeprom,do       ; no, set data bit =1
003B 092D           00181         call    BITOUT          ; transmit 1 bit to serial EE
003C 0370           00182         rlf     txbuf, F        ; rotate txbuf left
003D 02F1           00183         decfsz  count, F        ; all bits done?
003E 0A38           00184         goto    TXLP            ; no, do another bit
003F 0800           00185         retlw   0               ; yes, jump out
                    00186 ;
```


© 1997 Microchip Technology Inc.

```
                00187 ;*****************************************************************
                00188 ;                 POWER-UP ROUTINE
                00189 ;         This is the program entry point, which in this case simply
                00190 ;         sets the port_a I/O lines and directs control to the
                00191 ;         erase/write enable routine.
                00192 ;*****************************************************************
0040            00193 PWRUP
                00194 ;
0040 0C11       00195           movlw   b'00010001'
0041 0005       00196           tris    port_a          ; set RA0 as input, rest output
0042 0065       00197           clrf    port_a          ; all lines low
0043 0C80       00198           movlw   b'10000000'
0044 0006       00199           tris    port_b          ; set RB7 as input, rest output;
                00200 ;
                00201 ;                 Fall through and do the read
                00202 ;
                00203 ;*****************************************************************
                00204 ;         READ ROUTINE (Sequential read mode)
                00205 ;         This routine reads 8 consecutive addresses in
                00206 ;         sequential mode starting at address 0.  This
                00207 ;         program will repeat forever
                00208 ;*****************************************************************
0045            00209 READ
                00210 ;
0045 0C00       00211           movlw   .0              ; set starting address to 00
0046 002C       00212           movwf   addr            ;
0047 0C08       00213           movlw   .8              ; set number of addresses to
0048 0033       00214           movwf   bytcnt          ; read as 8
                00215                                   ;
0049 090E       00216           call    BSTART          ; generate the start bit
004A 0C02       00217           movlw   .2              ; set # bits to 2
004B 0032       00218           movwf   bits            ;
004C 0C80       00219           movlw   b'10000000'     ; get opcode (10b)
004D 0030       00220           movwf   txbuf           ; into output buffer
004E 0936       00221           call    TX              ; and transmit it
004F 0C08       00222           movlw   .8              ;
0050 0032       00223           movwf   bits            ; set number of bits to 8
0051 020C       00224           movf    addr,w          ; get the address
0052 0030       00225           movwf   txbuf           ; into the output buffer
0053 0936       00226           call    TX              ; and transmit it
                00227
0054 0921       00228 rbyte     call    RX              ; read the high byte
0055 020D       00229           movf    datai,w         ; move input data to w
0056 0037       00230           movwf   hbyte           ; xfer it to high byte
                00231
0057 0921       00232           call    RX              ; read the low byte
0058 020D       00233           movf    datai,w         ; move input data to w
0059 0037       00234           movwf   hbyte           ; xfer it to low byte
                00235
005A 02AC       00236           incf    addr, F         ; add 1 to the address
005B 02F3       00237           decfsz  bytcnt, F       ; have all bytes been read?
005C 0A54       00238           goto    rbyte           ; no, read another byte
                00239
005D 0486       00240           bcf     port_b,chpsel   ; yes,clear the chip select line
005E 0A45       00241           goto    READ            ; and start over
                00242       END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:    96
Program Memory Words Free:   416
```


© 1997 Microchip Technology Inc.

```
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN562

Using the Microchip Endurance Predictive Software

Authors: *Peter Sorrells*
Richard J. Fisher
David Wilkie
Microchip Technology Inc.

INTRODUCTION

Endurance, as it applies to nonvolatile EEPROM memory, refers to the number of times an individual memory cell can be erased and written. In this discussion, the term "ENDURANCE" will be used to describe the collective endurance of all memory cells in the EEPROM device (for a more detailed review of what EEPROM endurance is, refer to Microchip Technology Incorporated application notes AN536 in Serial EEPROM Tutorial Series Volume I, and AN537 in this tutorial series). Recent advances in process technology have made it possible to increase these limits and for Microchip to offer new concepts - Total Endurance™ and a split architectural design for variable endurance. These concepts lead to more reliable products with more bits per dice, such as the 24C32 and 24C65. It should be noted that this predictive model applies only to Microchip nonvolatile devices.

TOTAL ENDURANCE

When defining endurance, we need to look at a few common definitions and possible misconceptions. Endurance with respect to EEPROMs is defined in number of Erase/Write (E/W) Cycles and is the most common rating referred to when discussing or specifying endurance. E/W ratings are based on the environmental and operating conditions of voltage, temperature, cycling mode and rate (for each byte in the application not on the number of opcodes or control byte commands) and is never based on any read functions whether they be a data read or configuration read. If a part is rated at 100K E/W cycles, then each individual byte can be erased and written 100,000 times. This is probably the most common misinterpretation made by system designers. Endurance is thus an E/W application-specific reliability parameter. It is not a typical data sheet specification, such as a parametric AC/DC specification with benchmark standards for measurement. Microchip has done extensive predictive laboratory studies on its 2- and 3-wire Serial EEPROMs. These studies led to the concept of using the computer to predict the theoretical wear-out of the memory cell and ultimately to project the point in time of a product's life cycle when the first nonvolatile memory bit or periphery failure should occur. After many years of data collecting, predicting, and verifying the results, Microchip feels confident in publishing and offering for the general technical community this predictive model in the form of Windows®-based, IBM® PC-compatible software. Microchip has a patent pending on this predictive mathematical model.

TOTAL ENDURANCE PREDICTIVE SOFTWARE DESCRIPTION

The predictive software described here was originally developed as a tool for determining endurance levels of Microchip nonvolatile devices. Upon seeing the potential of the software as a design aid, it was decided to make a version that could be purchased by the engineering community. The benefit gained from this software is the ability to predict endurance capability of Microchip's EEPROM devices under various operating conditions. Prior to this tool becoming available, the only way to assemble this type of data would be to do extensive life testing in the target system.

The program uses an iterative statistical model developed by Microchip's physicists. The model was first used in a DOS-based text program as a proof of concept and for developing the database needed for such a tool (included on the program disk as enddos.exe). This model was then imported to a Windows-based software package with all the normal cut, paste, print, and viewing properties. The model actually operates as a mathematical function which is called from within the Windows Visual Basic shell and is passed all of the pertinent user application specific information. The model then, after calculating the essential data points, returns this information to the main program to be formatted and displayed both in a text and graph format.

7

© 1997 Microchip Technology Inc.

Example Application

Applying the predictive data to the high endurance (HE) block of the 24C65, using the 24LC04 which has similar characteristics, and assuming the following:

- a five-year life
- an expected E/W cycles of 10 times per day
- a function of 11 bytes

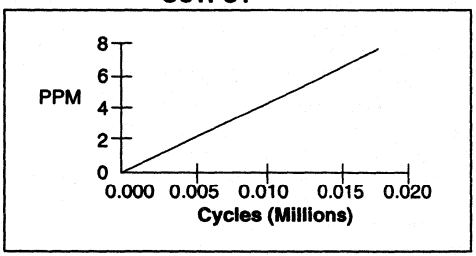
Operational specifications:

Device Data:	Input Parameter
Device	24C65 (24LC04B)
Voltage	5
Temperature	25°C
Bytes/Cycle	11
E/W/Day	10
App. Life (Yr.)	5
Cycling Mode	BYTE
Data Pattern	RANDOM

The 4K High Endurance block with 1 Million E/W cycles typical, in this application, should yield the following results:

FIT	1.0 PPM	8
Time	5.0 Write cycles	18,250

FIGURE 1: EXAMPLE OF GRAPHICAL OUTPUT



The results shown are predictive in nature and should reflect an accurate representation of the expected results. All operation parameters, along with the process technology, affect the effective endurance of a nonvolatile device. The voltage, temperature, cycles per, bytes per cycle, and even the number of times written per day (time between write cycles) all have an effect on the oxide breakdown or periphery failure rate of a particular nonvolatile process.

Endurance is not a well-defined concept within the semiconductor industry. The number of erase/write cycles which a particular EEPROM can endure is dependent not only upon the design of the device but also upon the application environment in which it is used. Therefore, blanket claims such as '1 million erase/write cycles typical' can only be validated based upon the specific parameters of each application. Yet until now, there has been no tool available for predicting the endurance of a particular EEPROM device within a set of application parameters. The trade-off analysis, previously required, can be painfully time-consuming and only marginally accurate without specific knowledge of the behavior of the device under different conditions of use.

The Microchip Total Endurance Software allows the designer to trade off voltage, temperature, write cycles, number of bytes written, number of writes per day, PPM and FIT rates, and years of use in order to optimize the system and accurately predict product lifetime and reliability.

The following is an example using the Endurance Software to aid in the design of an electronic phone book/ auto-dialer.

The auto-dialer may have new numbers added or changed several times per day; but how can the manufacturer specify the life of the unit, and at what rate of update of the phone numbers? First, the designer must make some assumptions. If we assume that the average user will change or add 50 phone numbers per day, and the manufacturer is willing to live with a 0.1% failure rate (1,000 PPM) after 10 years of use, then we have almost enough information to verify whether we are in the ball park given the physics of the EEPROM device which will store the numbers. We also need to know the operating voltage and temperature of the application; we will say that a 3.3V lithium button battery is powering the unit and the temperature range is limited to that for which the LCD display will function: 0°C to 70°C. End-of-life voltage for the battery is approximately 2.0V; assuming that the ASIC or microcontroller in the application will operate down to 2.5V, the EEPROM also has a 2.5V requirement. The designer would like to be able to store 100 phone numbers of 16 bytes each, which results in a 1.6K byte requirement for the Serial EEPROM. Because 1.6K bytes is equal to 12.8K bits, a 16K bit 2-wire Serial EEPROM will more than suffice. Specifically, Microchip's 24LC16B will operate down to 2.5V and even includes a write-protect feature which can be used to block inadvertent writes in a noisy environment.

Here is a summary of the application:

Device Data:	Input Parameters
Device	24LC16B
Voltage	2.5V - 3.3V
Temperature	0°C to 70°C (55°C typical)
Cycles per day	50
Bytes per cycle	16
Application life	10 years

© 1997 Microchip Technology Inc.

Once these values are entered into the Total Endurance program, it outputs the following:

Device Data:	Input Parameters
Device	24LC16B
Voltage	3.3
Temperature	55°C
Bytes/Cycle	16
E/W	50
App. Life (Yrs)	10
Cycling Mode	BYTE
Pulse Width (Ms)	N/A
Data Pattern	RANDOM

Device Data:	Output Parameters
FIT	19.1
PPM	1,670
Time	10.0
Write cycles	82,500

Both of the lists above were copied directly from the Total Endurance program output to the Microsoft Windows® clipboard and pasted into this document (the Total Endurance program has a handy menu click to make this easy).

Unfortunately for our designer, the desired 0.1% failure rate has almost doubled to 0.17% (1,670 PPM). But fortunately for the designer, the Total Endurance program makes trade-off analysis very simple and fast. At this point there are at least three options: (1) live with almost 2,000 PPM, or (2) look at the endurance plot and check whether there is a reasonable number of E/W cycles which will provide a 1,000 PPM failure rate, or (3) specify a PPM rate to the Total Endurance program and let it crank out the number of cycles it will take.

Below is the endurance plot, again pasted directly from the Total Endurance program:

You can see that by reducing the number of cycles from the 182,500 which resulted from our first trial to about 100,000, we can achieve a PPM rate of about 1,000 (0.1%). But how does 100,000 cycles translate into application life or cycles per day?

By switching the Total Endurance program mode to a PPM request mode instead of application life mode, we can query the program for this information. Let's ask it for the application life of the product given a 1,000 PPM failure rate. Here are the results:

Device Data:	Input Parameters
Device	24LC16B
Voltage	3.3
Temperature	55°C
Bytes/Cycle	16
E/W	50
PPM Level (Yrs)	1,000
Cycling Mode	BYTE
Pulse Width (Ms)	N/A
Data Pattern	RANDOM

Device Data:	Output Parameters
PPM	1,000
Time	6.52
Write cycles	119,000

Now we have some more options: (1) specify the product life at 5 years or (2) trade-off other parameters of the application such as voltage or temperature, or (3) decide which is more important a 10-year product lifetime, or the ability to change 50 numbers every single day. Maybe this analysis has caused our designer to reevaluate the 50 cycle-per-day requirement. Will the user really change or add that many numbers per day - half of the unit's total capacity? Maybe 20 or even 10 is a more practical figure. Realistically, a user may enter or change quite a few numbers the first week or two of the application, and after that the unit will be used mostly for reading and dialing numbers.

Changing the number of erase/write cycles to 20 per day gives us the following results:

Device Data:	Input Parameters
Device	24LC16B
Voltage	3.3
Temperature	55°C
Bytes/Cycle	16
E/W	20
PPM Level (Yrs)	1,000
Cycling Mode	BYTE
Pulse Width (Ms)	N/A
Data Pattern	RANDOM

Device Data:	Output Parameters
PPM	1,000
Time	16.3
Write cycles	109,000

© 1997 Microchip Technology Inc.

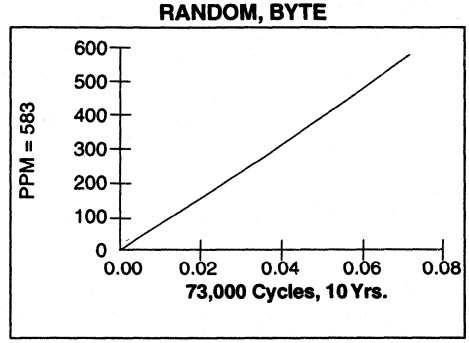
By reducing the number of cycles per day not only brought us back to a 10-year life, it gave us some reserve on that, too. Keeping all the other parameters the same and forcing a 10-year lifetime gives us the following final results:

Device Data:	Output Parameters
FIT	6.7
PPM	583
Time	10.0
Write cycles	73,000

The new PPM rate of 583 gives our triumphant designer more than 40% reserve on his PPM target of 1,000.

This example shows the significant reduction in time for design trade-off analysis and time-to-market which can be achieved with a useful tool like the Microchip Total Endurance Disk. In addition, it demonstrates the increase in robustness of the system design by providing known quantities and readily accessible handles to modify those quantities in the trade-off analysis. This tool can literally reduce weeks of effort into a few minutes of point and click.

FIGURE 2: 24LC16B, 55C, 3.3V, 16 BYTES, 20 CYCLES/DAY, RANDOM, BYTE



© 1997 Microchip Technology Inc.

Interfacing the 24LCXXB Serial EEPROMs to the PIC16C54

Author: *Bruce Negley*
Microchip Technology Inc.

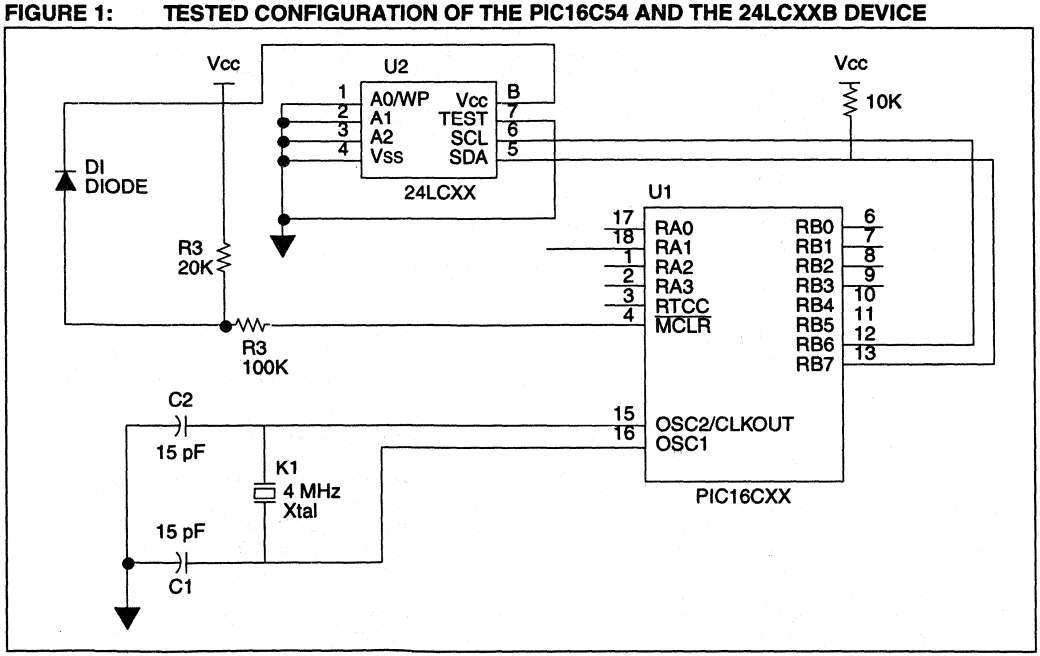
INTRODUCTION

The 24LCXXB Serial EEPROMs from Microchip Technology are I²C™ compatible, will operate in the standard 100 kHz and the 400 kHz Fast Mode. When using any serial protocol with a general microcontroller that does not have a dedicated protocol specific serial port, the designer must generate the specific code routines to accomplish the several memory access functions that the microcontroller will perform. The PIC16CXXX products from Microchip are both versatile and efficient to program. This application note is a series of stand-alone programs to perform the basic I²C interface functions on a PIC16C54 running at 4MHz in XT mode. This configuration will provide a 60 kHz serial bus rate. At this speed, the interface does not meet the 100kHz specification. If a higher clock rate is desired, the HS crystal oscillator must be used, which has a maximum frequency of 20MHz. NOTE THAT THE TIMING ROUTINES MUST BE REWRITTEN TO RUN AT THESE FASTER CLOCK RATES. The user must consult all applicable data sheets for design details. These programs have been fully tested and are being made available for general use. Figure 1 demonstrates the tested configuration of PIC16C54 and the 24LCXXB device.

This application note includes the following programs.

- 2-Wire Byte Read
- 2-Wire Byte Write
- 2-Wire Byte Write with Data Polling
- 2-Wire Page Write
- 2-Wire Sequential Read

FIGURE 1: TESTED CONFIGURATION OF THE PIC16C54 AND THE 24LCXXB DEVICE



7

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:2WBYTER.ASM

```
MPASM 01.40 Released          2WBYTER.ASM   1-16-1997  16:50:42         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54,c=132
                     00002 ;*****************************************************************
                     00003 ;       2-Wire Byte Read Program (118 bytes)
                     00004 ;
                     00005 ;       This program demonstrates how to interface a
                     00006 ;       Microchip PIC 16C54 to a 24LCXX Serial EE device
                     00007 ;       and perform a random read operation. This program
                     00008 ;       will read 8 consecutive addresses in the 'random
                     00009 ;       read' mode.  This entails setting the address pointer
                     00010 ;       before doing read command for every address.
                     00011 ;
                     00012 ;       Another, more efficient method of reading consecutive
                     00013 ;       addresses is the 'sequential read' mode.  This involves
                     00014 ;       sending the control byte and address for the first byte
                     00015 ;       to read, then continuing to provide clocks for the next
                     00016 ;       addresses.  The device will automatically increment the
                     00017 ;       address.  An example of the sequential read mode is
                     00018 ;       provided in the '2wseqr.asm' file.
                     00019 ;
                     00020 ;       Timing is based on using the PIC 16C54 in 'XT' mode
                     00021 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00022 ;       will be approximately 60 Khz for this setup.
                     00023 ;
                     00024 ;       As an option, the user may connect a LED to pin 18
                     00025 ;       on the PIC 16C54 (use about a 1K resistor in series) as an
                     00026 ;       acknowledge fail indicator.  This LED will come on if
                     00027 ;       the serial EE fails to acknowledge correctly.
                     00028 ;
                     00029 ;        PIC 16C54 to Serial EE Connections:
                     00030 ;
                     00031 ;        PIC 16C54       Serial EE
                     00032 ;       ------------    ------------
                     00033 ;       Pin 12 (RB6) -->  SCLK
                     00034 ;       Pin 13 (RB7) -->  SDATA
                     00035 ;
                     00036 ;       PIN 18 (RA1) -->  Acknowledge fail LED  (Optional)
                     00037 ;
                     00038 ;
                     00039 ;       Program:          2WBYTER.ASM
                     00040 ;       Revision Date:
                     00041 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00042 ;
                     00043 ;
                     00044 ;*****************************************************************
                     00045 ;           Register Definitions
                     00046 ;*****************************************************************
00000003             00047 status  equ     3h      ; status register
00000005             00048 port_a  equ     5h      ; port 5 (port a) used for LED display
00000006             00049 port_b  equ     6h      ; port 6 (port b) used for data and
                     00050                         ; clock lines
0000000A             00051 eeprom  equ    0ah      ; bit buffer
0000000B             00052 bycnt   equ    0bh      ; byte counter for read mode
0000000C             00053 addr    equ    0ch      ; address counter
0000000D             00054 datai   equ    0dh      ; data input register
```


© 1997 Microchip Technology Inc.

```
0000000E          00055 datao   equ    0eh     ; data output register
0000000F          00056 slave   equ    0fh     ; device address 1010xxx0)
00000010          00057 txbuf   equ    10h     ; transmit buffer
00000011          00058 count   equ    11h     ; bit counter
00000012          00059 bcount  equ    12h     ; byte counter
00000015          00060 loops   equ    15h     ; delay loop counter
00000016          00061 loops2  equ    16h     ; delay loop counter
                  00062
                  00063 ;*****************************************************
                  00064 ;           Bit Definitions
                  00065 ;*****************************************************
00000007          00066 di      equ    7       ; eeprom input bit
00000006          00067 do      equ    6       ; eeprom output bit
00000007          00068 sdata   equ    7       ; serial EE data line (port_b,pin 13)
00000006          00069 sclk    equ    6       ; serail EE clock line (port_b,pin 12)
00000001          00070 ackf    equ    1       ; acknowledge fail LED (port_a)
                  00071 ;*****************************************************
                  00072 ;
01FF              00073         org    01ffh   ; set reset vector
01FF 0A5E         00074         goto   PWRUP
0000              00075         org    000h    ;
0000 0A5E         00076         goto   PWRUP
                  00077 ;
                  00078 ;*************************************************************
                  00079 ;       Start Bit Subroutine
                  00080 ;       this routine generates a start bit
                  00081 ;       (Low going data line while clock is high)
                  00082 ;*************************************************************
                  00083 ;
0001              00084 BSTART
0001 05E6         00085         bsf    port_b,sdata   ; make sure data is high
0002 0C3F         00086         movlw  b'00111111'
0003 0006         00087         tris   port_b         ; set data and clock lines for output
0004 04C6         00088         bcf    port_b,sclk    ; make sure clock is low
0005 0000         00089         nop
0006 05C6         00090         bsf    port_b,sclk    ; set clock high
0007 0000         00091         nop
0008 0000         00092         nop
0009 0000         00093         nop
000A 0000         00094         nop
000B 0000         00095         nop
000C 04E6         00096         bcf    port_b,sdata   ; data line goes low during
                  00097                                ; high clock for start bit
000D 0000         00098         nop
000E 0000         00099         nop
000F 0000         00100         nop
0010 0000         00101         nop
0011 0000         00102         nop                   ; timing adjustment
0012 04C6         00103         bcf    port_b,sclk    ; start clock train
0013 0000         00104         nop
0014 0000         00105         nop
0015 0800         00106         retlw  0
                  00107         ;
                  00108 ;*************************************************************
                  00109 ;       Stop Bit Subroutine
                  00110 ;       This routine generates a stop bit
                  00111 ;       (High going data line while clock is high)
                  00112 ;*************************************************************
0016              00113 BSTOP
0016 04E6         00114         bcf    port_b,sdata   ; make sure data line is low
0017 0C3F         00115         movlw  b'00111111'    ;
0018 0006         00116         tris   port_b         ; set data/clock lines as outputs
0019 04E6         00117         bcf    port_b,sdata   ; make sure data line is low
001A 0000         00118         nop
001B 0000         00119         nop
001C 0000         00120         nop
```


© 1997 Microchip Technology Inc.

```
001D 05C6          00121          bsf     port_b,sclk     ; set clock high
001E 0000          00122          nop
001F 0000          00123          nop
0020 0000          00124          nop
0021 05E6          00125          bsf     port_b,sdata    ; data goes high while clock high
                   00126                                  ; for stop bit
0022 0000          00127          nop
0023 0000          00128          nop
0024 04C6          00129          bcf     port_b,sclk     ; set clock low again
0025 0000          00130          nop
0026 0000          00131          nop
0027 0000          00132          nop
0028 0800          00133          retlw   0
                   00134 ;
                   00135 ;*****************************************************************
                   00136 ;       BITOUT routine takes the bit of data in 'do' and
                   00137 ;       transmits it to the serial EE device
                   00138 ;*****************************************************************
0029               00139 BITOUT
0029 0C3F          00140          movlw   b'00111111'     ; set data,clock as outputs
002A 0006          00141          tris    port_b
002B 07CA          00142          btfss   eeprom,do       ; check for state of data bit to xmit
002C 0A2F          00143          goto    bitlow          ; low? go set data line low
002D 05E6          00144          bsf     port_b,sdata    ; high? set data line high
002E 0A30          00145          goto    clkout          ; go toggle the clock
                   00146
002F 04E6          00147 bitlow   bcf     port_b,sdata    ; output a low bit
0030 05C6          00148 clkout   bsf     port_b,sclk     ; set clock line high
0031 0000          00149          nop
0032 0000          00150          nop
0033 0000          00151          nop
0034 0000          00152          nop
0035 04C6          00153          bcf     port_b,sclk     ; return clock line low
0036 0800          00154          retlw   0
                   00155 ;
                   00156 ;*****************************************************************
                   00157 ;       BITIN routine reads one bit of data from the
                   00158 ;       serial EE device and stores it in the bit 'di'
                   00159 ;*****************************************************************
0037               00160 BITIN
0037 05EA          00161          bsf     eeprom,di       ; assume input bit is high
0038 0CBF          00162          movlw   b'10111111'     ; make sdata an input line
0039 0006          00163          tris    port_b
003A 05C6          00164          bsf     port_b,sclk     ; set clock line high
003B 0000          00165          nop                     ; just sit here a sec
003C 0000          00166          nop
003D 0000          00167          nop
003E 0000          00168          nop
003F 0000          00169          nop                     ;
0040 07E6          00170          btfss   port_b,sdata    ; read the data bit
0041 04EA          00171          bcf     eeprom,di       ; input bit was low, set 'di' accordingly
0042 04C6          00172          bcf     port_b,sclk     ; set clock line low
                   00173                                  ;
0043 0800          00174          retlw   0               ;
                   00175 ;
                   00176 ;*****************************************************************
                   00177 ;       Transmit Data Subroutine
                   00178 ;       This routine takes the byte of data stored in the
                   00179 ;       'datao' register and transmits it to the serial EE device.
                   00180 ;       It will then send 1 more clock to the serial EE for the
                   00181 ;       acknowledge bit.  If the ack bit from the part was low
                   00182 ;       then the transmission was sucessful.  If it is high, then
                   00183 ;       the device did not send a proper ack bit and the ack
                   00184 ;       fail LED will be turned on.
                   00185 ;*****************************************************************
0044               00186 TX
```


© 1997 Microchip Technology Inc.

```
0044 0C08          00187          movlw   .8
0045 0031          00188          movwf   count          ; set the #bits to 8
                   00189          ;
0046               00190 TXLP
0046 04CA          00191          bcf     eeprom,do      ; assume bit out is low
0047 06F0          00192          btfsc   txbuf,7        ; is bit out really low?
0048 05CA          00193          bsf     eeprom,do      ; no, set it high
0049 0929          00194          call    BITOUT         ; send the bit to serial EE
004A 0370          00195          rlf     txbuf, F       ; rotate txbuf left
004B 02F1          00196          decfsz  count, F       ; 8 bits done?
004C 0A46          00197          goto    TXLP           ; no - go again
004D 0937          00198          call    BITIN          ; read ack bit
004E 06EA          00199          btfsc   eeprom,di      ; check ack bit
004F 0525          00200          bsf     port_a,ackf    ; set acknowledge fail LED if the
                   00201                                 ; device did not pull data low
                   00202          ;
0050 0800          00203          retlw   0
                   00204 ;
                   00205 ;*****************************************************************
                   00206 ;       Receive data routine
                   00207 ;       This routine reads one byte of data from the part
                   00208 ;       into the 'datai' register.  It then sends a high
                   00209 ;       ack bit to indicate that no more data is to be read
                   00210 ;*****************************************************************
0051               00211 RX
0051 006D          00212          clrf    datai          ; clear input buffer
0052 0C08          00213          movlw   .8             ; set # bits to 8
0053 0031          00214          movwf   count
0054 0403          00215          bcf     status,0       ; make sure carry bit is low
0055 036D          00216 RXLP     rlf     datai, F       ; rotate datai 1 bit left
0056 0937          00217          call    BITIN          ; read a bit
0057 06EA          00218          btfsc   eeprom,di
0058 050D          00219          bsf     datai,0        ; set bit 0 if necessary
0059 02F1          00220          decfsz  count, F       ; 8 bits done?
005A 0A55          00221          goto    RXLP           ; no, do another
005B 05CA          00222          bsf     eeprom,do      ; set ack bit = 1
005C 0929          00223          call    BITOUT         ; to finish transmission
005D 0800          00224          retlw   0
                   00225 ;
                   00226 ;*****************************************************************
                   00227 ;       Power up routine
                   00228 ;       This is the program entry point.  I/O line status is
                   00229 ;       set for port A here.
                   00230 ;*****************************************************************
005E               00231 PWRUP
005E 0C00          00232           movlw  b'00000000'
005F 0005          00233           tris   port_a          ; set port A as all output
0060 0065          00234           clrf   port_a          ; all output lines low
0061 0A62          00235           goto   READ
                   00236 ;
                   00237 ;*****************************************************************
                   00238 ;       READ (read routine)
                   00239 ;       This routine reads 8 consecutive addresses of the
                   00240 ;       serial EE device starting at address 00 in the
                   00241 ;       random access mode (non-sequential read). Reading
                   00242 ;       the device using the random access mode
                   00243 ;       requires that the address pointer be set for every
                   00244 ;       byte before the read takes place. The address pointer
                   00245 ;       is set by sending a 'write mode' control byte with the
                   00246 ;       address of the location to read.
                   00247 ;*****************************************************************
0062               00248 READ
                   00249 ;
0062 0425          00250          bcf     port_a,ackf ; clear the ack fail LED if on
0063 0C08          00251          movlw   .8          ; set number of bytes to read as 8
0064 0032          00252          movwf   bcount      ;
```

7

```
0065 006C           00253          clrf    addr          ; set starting address to 00
                    00254                                ;
0066 0901           00255 rbyte    call    BSTART        ; generate start bit
                    00256                                ;
                    00257                                ; now send the write control byte and
                    00258                                ; address to set the pointer
                    00259                                ;
0067 0CA0           00260          movlw   b'10100000'   ; get slave address (write mode)
0068 0030           00261          movwf   txbuf         ; into transmit buffer
0069 0944           00262          call    TX            ; and send it
006A 020C           00263          movf    addr,w        ; get word address
006B 0030           00264          movwf   txbuf         ; into transmit buffer
006C 0944           00265          call    TX            ; and send it
                    00266                                ;
                    00267                                ; now read one byte from the part
                    00268                                ;
006D 0901           00269          call    BSTART        ; generate start bit
006E 0CA1           00270          movlw   b'10100001'   ; get slave address and read mode
006F 0030           00271          movwf   txbuf         ; into transmit buffer
0070 0944           00272          call    TX            ; and transmit it
0071 0951           00273          call    RX            ; read 1 byte from serial EE
0072 0916           00274          call    BSTOP         ; send stop bit to end transmission
0073 02AC           00275          incf    addr, F       ; add 1 to address counter
0074 02F2           00276          decfsz  bcount, F     ; yes, are all 8 bytes read?
0075 0A66           00277          goto    rbyte         ; no, do another byte
0076 0A62           00278          goto    READ          ; yes, start all over
                    00279 ;
                    00280      END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXX---------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   120
Program Memory Words Free:   392


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B:2WBYTEW.ASM

```
MPASM 01.40 Released          2WBYTEW.ASM   1-16-1997  16:51:05         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54,c=132
                     00002 ;*****************************************************************
                     00003 ;       2-Wire Byte Write Program (123 bytes)
                     00004 ;
                     00005 ;       This program demonstrates how to interface a
                     00006 ;       Microchip PIC 16C54 to a 24LCXX Serial EE device
                     00007 ;       and perform a byte write operation on 8 consecutive
                     00008 ;       addresses.
                     00009 ;
                     00010 ;       This routine waits approximately 10mS for the write
                     00011 ;       cycle time, which is enough time for any of the
                     00012 ;       24LCxx devices to complete a write.  A more efficient
                     00013 ;       method of determining when the write cycle is complete
                     00014 ;       is called "data polling."  The data polling method is
                     00015 ;       explained in the program "2wdpoll.asm."  That
                     00016 ;       particular program uses data polling in a byte write
                     00017 ;       mode but it can be used in exactly the same way for
                     00018 ;       a page mode write.
                     00019 ;
                     00020 ;       As an option, the user may connect a LED to pin 18
                     00021 ;       on the PIC 16C54 (use about a 1K resistor in series) as an
                     00022 ;       acknowledge fail indicator.  This LED will come on if
                     00023 ;       the serial EE fails to acknowledge correctly.
                     00024 ;
                     00025 ;       Timing is based on using the PIC 16C54 in 'XT' mode
                     00026 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00027 ;       will be approximately 60 Khz for this setup.
                     00028 ;
                     00029 ;        PIC 16C54 to Serial EE Connections:
                     00030 ;
                     00031 ;        PIC 16C54         Serial EE
                     00032 ;       ------------     ------------
                     00033 ;       Pin 12 (RB6) -->  SCLK
                     00034 ;       Pin 13 (RB7) -->  SDATA
                     00035 ;
                     00036 ;       PIN 18 (RA1) -->  Acknowledge fail LED  (Optional)
                     00037 ;
                     00038 ;
                     00039 ;       Program:          2WBYTEW.ASM
                     00040 ;       Revision Date:
                     00041 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00042 ;
                     00043 ;*****************************************************************
                     00044 ;           Register Definitions
                     00045 ;*****************************************************************
00000005             00046 port_a  equ     5h      ; port 5 (port_a) used for LEDs
00000006             00047 port_b  equ     6h      ; port 6 (port b) used for data and
                     00048                         ; clock lines
0000000A             00049 eeprom  equ     0ah     ; bit buffer
0000000B             00050 bycnt   equ     0bh     ; byte counter for read mode
0000000C             00051 addr    equ     0ch     ; address counter
0000000D             00052 datai   equ     0dh     ; data input register
0000000E             00053 datao   equ     0eh     ; data output register
0000000F             00054 slave   equ     0fh     ; device address
```

```
                    00055                           ; (1010xxx0)
  00000010          00056 txbuf   equ     10h       ; transmit buffer
  00000011          00057 count   equ     11h       ; bit counter
  00000012          00058 bcount  equ     12h       ; byte counter
  00000015          00059 loops   equ     15h       ; delay loop counter
  00000016          00060 loops2  equ     16h       ; delay loop counter
                    00061 ;*****************************************************
                    00062 ;           Bit Definitions
                    00063 ;*****************************************************
  00000007          00064 di      equ     7         ; eeprom input bit
  00000006          00065 do      equ     6         ; eeprom output bit
  00000007          00066 sdata   equ     7         ; serial EE data line (port_b,pin 13)
  00000006          00067 sclk    equ     6         ; serail EE clock line (port_b,pin 12)
  00000001          00068 ackf    equ     1         ; acknowledge fail LED (port_a,pin 18)
                    00069 ;*****************************************************
                    00070 ;
                    00071 ;
01FF                00072         org     01ffh     ; set reset vector
01FF 0A5E           00073         goto    PWRUP
0000                00074         org     000h      ;
0000 0A5E           00075         goto    PWRUP
                    00076 ;
                    00077 ;*****************************************************
                    00078 ;       DELAY ROUTINE
                    00079 ;       This routine takes the value in 'loops'
                    00080 ;       and multiplies it times 1 millisecond to
                    00081 ;       determine delay time.
                    00082 ;*****************************************************
0001                00083 WAIT
                    00084 ;
0001 0C6E           00085 top       movlw   .110            ; timing adjustment variable
0002 0036           00086           movwf   loops2
0003 0000           00087 top2      nop                     ; sit and wait
0004 0000           00088           nop
0005 0000           00089           nop
0006 0000           00090           nop
0007 0000           00091           nop
0008 0000           00092           nop
0009 02F6           00093           decfsz  loops2, F       ; inner loops complete?
000A 0A03           00094           goto    top2            ; no, go again
                    00095                                   ;
000B 02F5           00096           decfsz  loops, F        ; outer loops complete?
000C 0A01           00097           goto    top             ; no, go again
000D 0800           00098           retlw   0               ; yes, return from sub
                    00099 ;
                    00100 ;*****************************************************
                    00101 ;       Start Bit Subroutine
                    00102 ;       this routine generates a start bit
                    00103 ;       (Low going data line while clock is high)
                    00104 ;*****************************************************
                    00105 ;
000E                00106 BSTART
000E 05E6           00107           bsf     port_b,sdata    ; make sure data is high
000F 0C3F           00108           movlw   b'00111111'
0010 0006           00109           tris    port_b          ; set data and clock lines for output
0011 04C6           00110           bcf     port_b,sclk     ; make sure clock is low
0012 0000           00111           nop
0013 05C6           00112           bsf     port_b,sclk     ; set clock high
0014 0000           00113           nop
0015 0000           00114           nop
0016 0000           00115           nop
0017 0000           00116           nop
0018 0000           00117           nop
0019 04E6           00118           bcf     port_b,sdata    ; data line goes low during
                    00119                                   ; high clock for start bit
001A 0000           00120           nop
```


© 1997 Microchip Technology Inc.

```
001B 0000           00121          nop
001C 0000           00122          nop
001D 0000           00123          nop
001E 0000           00124          nop                     ; timing adjustment
001F 04C6           00125          bcf     port_b,sclk     ; start clock train
0020 0000           00126          nop
0021 0000           00127          nop
0022 0800           00128          retlw   0
                    00129          ;
                    00130          ;    End of Subroutine
                    00131 ;*******************************************************
                    00132 ;       Stop Bit Subroutine
                    00133 ;       This routine generates a stop bit
                    00134 ;       (High going data line while clock is high)
                    00135 ;*******************************************************
0023                00136 BSTOP
0023 0C3F           00137          movlw   b'00111111'     ;
0024 0006           00138          tris    port_b          ; set data/clock lines as outputs
0025 04E6           00139          bcf     port_b,sdata    ; make sure data line is low
0026 0000           00140          nop
0027 0000           00141          nop
0028 0000           00142          nop
0029 05C6           00143          bsf     port_b,sclk     ; set clock high
002A 0000           00144          nop
002B 0000           00145          nop
002C 0000           00146          nop
002D 05E6           00147          bsf     port_b,sdata    ; data goes high while clock high
                    00148                                  ; for stop bit
002E 0000           00149          nop
002F 0000           00150          nop
0030 04C6           00151          bcf     port_b,sclk     ; set clock low again
0031 0000           00152          nop
0032 0000           00153          nop
0033 0000           00154          nop
0034 0800           00155          retlw   0
                    00156 ;
                    00157 ;           End of Subroutine
                    00158 ;*******************************************************
                    00159 ;       BITOUT routine takes one bit of data in 'do' and
                    00160 ;       transmits it to the serial EE device
                    00161 ;*******************************************************
0035                00162 BITOUT
0035 0C3F           00163          movlw   b'00111111'     ; set data,clock as outputs
0036 0006           00164          tris    port_b
0037 07CA           00165          btfss   eeprom,do       ; check for state of data bit to xmit
0038 0A3B           00166          goto    bitlow          ;
0039 05E6           00167          bsf     port_b,sdata    ; set data line high
003A 0A3C           00168          goto    clkout          ; go toggle the clock
                    00169
003B 04E6           00170 bitlow   bcf     port_b,sdata    ; output a low bit
003C 05C6           00171 clkout   bsf     port_b,sclk     ; set clock line high
003D 0000           00172          nop
003E 0000           00173          nop
003F 0000           00174          nop
0040 0000           00175          nop
0041 04C6           00176          bcf     port_b,sclk     ; return clock line low
0042 0800           00177          retlw   0
                    00178 ;
                    00179 ;                 End of Subroutine
                    00180 ;
                    00181 ;*******************************************************
                    00182 ;       BITIN routine reads one bit of data from the
                    00183 ;       serial EE device and stores it in 'di'
                    00184 ;*******************************************************
0043                00185 BITIN
0043 05EA           00186          bsf     eeprom,di       ; assume input bit is high
```

7

```
0044 0CBF           00187          movlw   b'10111111'    ; make sdata an input line
0045 0006           00188          tris    port_b
0046 05E6           00189          bsf     port_b,sdata   ; set sdata line for input
0047 05C6           00190          bsf     port_b,sclk    ; set clock line high
0048 0000           00191          nop                    ; just sit here a sec
0049 0000           00192          nop
004A 0000           00193          nop
004B 0000           00194          nop
004C 0000           00195          nop                    ;
004D 07E6           00196          btfss   port_b,sdata   ; read the data bit
004E 04EA           00197          bcf     eeprom,di      ; input bit was low
004F 04C6           00198          bcf     port_b,sclk    ; set clock line low
                    00199                                 ;
0050 0800           00200          retlw   0              ;
                    00201 ;
                    00202 ;*****************************************************************
                    00203 ;       Transmit Data Subroutine
                    00204 ;       This routine takes the byte of data stored in the
                    00205 ;       'datao' register and transmits it to the serial EE device.
                    00206 ;       It will then send 1 more clock to the serial EE for the
                    00207 ;       acknowledge bit.  If the ack bit from the part was low
                    00208 ;       then the transmission was sucessful.  If it is high, then
                    00209 ;       the device did not send a proper ack bit and the ack
                    00210 ;       fail LED will be turned on.
                    00211 ;*****************************************************************
0051                00212 TX
0051 0C08           00213          movlw   .8
0052 0031           00214          movwf   count          ; set the #bits to 8
                    00215          ;
0053                00216 TXLP
0053 04CA           00217          bcf     eeprom,do      ; assume bit out is low
0054 06F0           00218          btfsc   txbuf,7        ; is bit out really low?
0055 05CA           00219          bsf     eeprom,do      ; otherwise data bit =1
0056 0935           00220          call    BITOUT         ; serial data out
0057 0370           00221          rlf     txbuf, F       ; rotate txbuf left
0058 02F1           00222          decfsz  count, F       ; 8 bits done?
0059 0A53           00223          goto    TXLP           ; no - go again
005A 0943           00224          call    BITIN          ; read ack bit
005B 06EA           00225          btfsc   eeprom,di      ; check ack bit
005C 0525           00226          bsf     port_a,ackf    ; set acknowledge fail LED if the
                    00227          ;
005D 0800           00228          retlw   0
                    00229 ;
                    00230 ;*****************************************************************
                    00231 ;       Power up routine
                    00232 ;       This is the program entry point, which in this case simply
                    00233 ;       sets the port_a I/O lines and directs control to the
                    00234 ;       byte write routine.
                    00235 ;*****************************************************************
005E                00236 PWRUP
005E 0C00           00237           movlw  b'00000000'
005F 0005           00238           tris   port_a         ; set port A as all output
0060 0065           00239           clrf   port_a         ; all output lines low
0061 0A62           00240           goto   WRBYTE
                    00241
                    00242 ;*****************************************************************
                    00243 ;       Byte Write Routine
                    00244 ;       This routine writes the data in "datao" to
                    00245 ;       8 consecutive bytes in the serial EE device starting
                    00246 ;       at address 00.  This routine waits 10mS after every
                    00247 ;       byte to give the device time to do the write.  This
                    00248 ;       program repeats forever.
                    00249 ;*****************************************************************
                    00250 ;
0062                00251 WRBYTE
                    00252 ;
```


© 1997 Microchip Technology Inc.

```
0062 0065           00253          clrf    port_a       ; clear all LEDs
0063 0CA0           00254          movlw   b'10100000'  ; set slave address and write mode
0064 002F           00255          movwf   slave
0065 0C55           00256          movlw   b'01010101'  ; set data to write as 55h
0066 002E           00257          movwf   datao
                    00258 ;
0067 0C08           00259          movlw   .8           ; set number of bytes
0068 0032           00260          movwf   bcount       ; to write as to 8
0069 006C           00261          clrf    addr         ; set starting address to 00
                    00262                               ;
006A 090E           00263 byte     call    BSTART       ; generate start bit
006B 020F           00264          movf    slave,w      ; move slave address
006C 0030           00265          movwf   txbuf        ; into transmit buffer
006D 0951           00266          call    TX           ; and send it
006E 020C           00267          movf    addr,w       ; move word address
006F 0030           00268          movwf   txbuf        ; into transmit buffer
0070 0951           00269          call    TX           ; and send it
0071 020E           00270          movf    datao,w      ; move data byte
0072 0030           00271          movwf   txbuf        ; to tranmit buffer
0073 0951           00272          call    TX           ; and transmit it
0074 0923           00273          call    BSTOP        ; generate stop bit
                    00274                               ;
0075 0C0A           00275          movlw   .10
0076 0035           00276          movwf   loops        ; set delay time to give
0077 0901           00277          call    WAIT         ; 10 ms wait after every byte
0078 02AC           00278          incf    addr, F      ; add 1 to address counter
0079 02F2           00279          decfsz  bcount, F    ; all 8 bytes written?
007A 0A6A           00280          goto    byte         ; no, do another byte
                    00281                               ;
007B 0A62           00282          goto    WRBYTE       ; start over
                    00283 ;
                    00284 ;
                    00285      END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXX----
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   125
Program Memory Words Free:   387


Errors   :     0
Warnings :     0 reported,      0 suppressed
Messages :     0 reported,      0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX C:2WDPOLL.ASM

```
MPASM 01.40 Released          2WDPOLL.ASM   1-16-1997  16:51:28         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54,c=132
                     00002 ;*****************************************************************
                     00003 ;       2-Wire Byte Write With Data Polling Program (129 bytes)
                     00004 ;
                     00005 ;       This program demonstrates how to interface a
                     00006 ;       Microchip PIC 16C54 to a 24LCXX Serial EE device.
                     00007 ;       This program performs a byte write operation on 8
                     00008 ;       consecutive addresses using the data polling method
                     00009 ;       to determine when the write cycle is complete.
                     00010 ;
                     00011 ;       When writing to a serial E≤ device, there are 2
                     00012 ;       ways to handle the internal timed write cycle
                     00013 ;       time of the device.  The simplest method is
                     00014 ;       simply to wait until the maximum cycle time
                     00015 ;       is exceeded before attempting another command.
                     00016 ;       The other, more efficient method is known as "data
                     00017 ;       polling."  Data polling is done by sending a start
                     00018 ;       bit and control byte to the part after the write
                     00019 ;       cycle has been initiated by a stop bit.  If the ack bit
                     00020 ;       is low, then the device is through writing, otherwise
                     00021 ;       the the sequence is repeated.  If no low acknowledge
                     00022 ;       is found within 40 attempts (about 10 milliseconds)
                     00023 ;       then the routine times out and sets the timeout
                     00024 ;       LED (pin 1) high.
                     00025 ;
                     00026 ;       As an option, the user can connect a LED to pin 1
                     00027 ;       on the PIC 16C54 (use about a 1K resistor in series) as a
                     00028 ;       timeout indicator.  This LED will come on if the
                     00029 ;       data polling is unsuccessful and the device being
                     00030 ;       programmed does not respond.  This can be tested by
                     00031 ;       simply running the program with no part in the socket.
                     00032 ;
                     00033 ;       Timing is based on using the PIC 16C54 in 'XT' mode
                     00034 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00035 ;       will be approximately 40 Khz for this setup.
                     00036 ;
                     00037 ;        PIC 16C54 to Serial EE Connections:
                     00038 ;
                     00039 ;        PIC 16C54          Serial EE
                     00040 ;       ------------    ------------
                     00041 ;       Pin 12 (RB6) -->  SCLK
                     00042 ;       Pin 13 (RB7) -->  SDATA
                     00043 ;
                     00044 ;       Pin 1  (RA2) -->  Write cycle timeout fail LED (optional)
                     00045 ;
                     00046 ;
                     00047 ;       Program:          2WDPOLL.ASM
                     00048 ;       Revision Date:
                     00049 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00050 ;
                     00051 ;*****************************************************************
                     00052 ;           Register Definitions
                     00053 ;*****************************************************************
  00000005           00054 port_a  equ     5h      ; port 5 (port_a) used LEDs
```


© 1997 Microchip Technology Inc.

```
00000006          00055 port_b  equ     6h      ; port 6 (port_b) used for data and
                  00056                         ; clock lines
0000000A          00057 eeprom  equ     0ah     ; bit buffer
0000000B          00058 bycnt   equ     0bh     ; byte counter for read mode
0000000C          00059 addr    equ     0ch     ; address counter
0000000D          00060 datai   equ     0dh     ; data input register
0000000E          00061 datao   equ     0eh     ; data output register
0000000F          00062 slave   equ     0fh     ; device address (1010xxx0)
00000010          00063 txbuf   equ     10h     ; transmit buffer
00000011          00064 count   equ     11h     ; bit counter
00000012          00065 bcount  equ     12h     ; byte counter
00000015          00066 loops   equ     15h     ; delay loop counter
00000016          00067 loops2  equ     16h     ; delay loop counter
00000017          00068 pollcnt equ     17h     ; data polling counter
                  00069 ;*****************************************************
                  00070 ;           Bit Definitions
                  00071 ;*****************************************************
00000007          00072 di      equ     7       ; eeprom input bit
00000006          00073 do      equ     6       ; eeprom output bit
00000007          00074 sdata   equ     7       ; serial EE data line (port_b,pin 13)
00000006          00075 sclk    equ     6       ; serail EE clock line (port_b,pin 12)
00000002          00076 timeout equ     2       ; write cycle timeout fail LED, port_a (pin 1)
00000001          00077 ackf    equ     1       ; acknowledge fail LED, port_a (pin 18)
                  00078 ;*****************************************************
                  00079 ;
01FF              00080         org     01ffh   ; set reset vector
01FF 0A5C         00081         goto    PWRUP
0000              00082         org     000h    ;
0000 0A5C         00083         goto    PWRUP
                  00084 ;
                  00085 ;*****************************************************
                  00086 ;       DELAY ROUTINE
                  00087 ;       This routine takes the value in 'loops'
                  00088 ;       and multiplies it times 1 millisecond to
                  00089 ;       determine delay time.
                  00090 ;*****************************************************
0001              00091 WAIT
                  00092 ;
0001 0C6E         00093 top      movlw   .110           ; timing adjustment variable
0002 0036         00094          movwf   loops2
0003 0000         00095 top2     nop                    ; sit and wait
0004 0000         00096          nop
0005 0000         00097          nop
0006 0000         00098          nop
0007 0000         00099          nop
0008 0000         00100          nop
0009 02F6         00101          decfsz  loops2, F      ; inner loops complete?
000A 0A03         00102          goto    top2           ; no, go again
                  00103                                 ;
000B 02F5         00104          decfsz  loops, F       ; outer loops complete?
000C 0A01         00105          goto    top            ; no, go again
000D 0800         00106          retlw   0              ; yes, return from sub
                  00107 ;
                  00108 ;*****************************************************
                  00109 ;       Start Bit Subroutine
                  00110 ;       this routine generates a start bit
                  00111 ;       (Low going data line while clock is high)
                  00112 ;*****************************************************
                  00113 ;
000E              00114 BSTART
000E 05E6         00115          bsf     port_b,sdata   ; make sure data is high
000F 0C3F         00116          movlw   b'00111111'
0010 0006         00117          tris    port_b         ; set data and clock lines for output
0011 04C6         00118          bcf     port_b,sclk    ; make sure clock is low
0012 0000         00119          nop
0013 05C6         00120          bsf     port_b,sclk    ; set clock high
```


© 1997 Microchip Technology Inc.

```
0014 0000           00121          nop
0015 0000           00122          nop
0016 0000           00123          nop
0017 0000           00124          nop
0018 0000           00125          nop
0019 04E6           00126          bcf     port_b,sdata    ; data line goes low during
                    00127                                  ; high clock for start bit
001A 0000           00128          nop
001B 0000           00129          nop
001C 0000           00130          nop
001D 0000           00131          nop
001E 0000           00132          nop                    ; timing adjustment
001F 04C6           00133          bcf     port_b,sclk    ; start clock train
0020 0000           00134          nop
0021 0000           00135          nop
0022 0800           00136          retlw   0
                    00137          ;
                    00138 ;*********************************************************
                    00139 ;       Stop Bit Subroutine
                    00140 ;       This routine generates a stop bit
                    00141 ;       (High going data line while clock is high)
                    00142 ;*********************************************************
0023                00143 BSTOP
0023 0C3F           00144          movlw   b'00111111'    ;
0024 0006           00145          tris    port_b         ; set data/clock lines as outputs
0025 04E6           00146          bcf     port_b,sdata   ; make sure data line is low
0026 0000           00147          nop
0027 0000           00148          nop
0028 0000           00149          nop
0029 05C6           00150          bsf     port_b,sclk    ; set clock high
002A 0000           00151          nop
002B 0000           00152          nop
002C 0000           00153          nop
002D 05E6           00154          bsf     port_b,sdata   ; data goes high while clock high
                    00155                                 ; for stop bit
002E 0000           00156          nop
002F 0000           00157          nop
0030 04C6           00158          bcf     port_b,sclk    ; set clock low again
0031 0000           00159          nop
0032 0000           00160          nop
0033 0000           00161          nop
0034 0800           00162          retlw   0
                    00163 ;
                    00164 ;*********************************************************
                    00165 ;       BITOUT routine takes one bit of data in 'do' and
                    00166 ;       transmits it to the serial EE device
                    00167 ;*********************************************************
0035                00168 BITOUT
0035 0C3F           00169          movlw   b'00111111'    ; set data,clock as outputs
0036 0006           00170          tris    port_b
0037 07CA           00171          btfss   eeprom,do      ; check for state of data bit to xmit
0038 0A3B           00172          goto    bitlow         ;
0039 05E6           00173          bsf     port_b,sdata   ; set data line high
003A 0A3C           00174          goto    clkout         ; go toggle the clock
                    00175                                 ;
003B 04E6           00176 bitlow   bcf     port_b,sdata   ; output a low bit
003C 05C6           00177 clkout   bsf     port_b,sclk    ; set clock line high
003D 0000           00178          nop
003E 0000           00179          nop
003F 0000           00180          nop
0040 0000           00181          nop
0041 04C6           00182          bcf     port_b,sclk    ; return clock line low
0042 0800           00183          retlw   0
                    00184 ;
                    00185 ;                   End of Subroutine
                    00186 ;
```


© 1997 Microchip Technology Inc.

```
                00187 ;*****************************************************************
                00188 ;       BITIN routine reads one bit of data from the
                00189 ;       serial EE device and stores it in 'di'
                00190 ;*****************************************************************
0043            00191 BITIN
0043 05EA       00192           bsf     eeprom,di       ; assume input bit is high
0044 0CBF       00193           movlw   b'10111111'     ; make sdata an input line
0045 0006       00194           tris    port_b
0046 05E6       00195           bsf     port_b,sdata    ; set sdata line for input
0047 05C6       00196           bsf     port_b,sclk     ; set clock line high
0048 0000       00197           nop                     ; just sit here a sec
0049 0000       00198           nop
004A 0000       00199           nop
004B 0000       00200           nop
004C 0000       00201           nop                     ;
004D 07E6       00202           btfss   port_b,sdata    ; read the data bit
004E 04EA       00203           bcf     eeprom,di       ; input bit was low
004F 04C6       00204           bcf     port_b,sclk     ; set clock line low
                00205                                   ;
0050 0800       00206           retlw   0               ;
                00207 ;
                00208 ;*****************************************************************
                00209 ;       Transmit Data Subroutine
                00210 ;       This routine takes the byte of data stored in the
                00211 ;       'datao' register and transmits it to the serial EE device.
                00212 ;       It will then send 1 more clock to the serial EE for the
                00213 ;       acknowledge bit.  If the ack bit from the part was low
                00214 ;       then the transmission was sucessful.  If it is high, then
                00215 ;       the device did not send a proper ack bit and the ack
                00216 ;       fail LED will be turned on.
                00217 ;*****************************************************************
0051            00218 TX
0051 0C08       00219           movlw   .8
0052 0031       00220           movwf   count           ; set the #bits to 8
                00221           ;
0053            00222 TXLP
0053 04CA       00223           bcf     eeprom,do       ; assume bit out is low
0054 06F0       00224           btfsc   txbuf,7         ; is bit out really low?
0055 05CA       00225           bsf     eeprom,do       ; otherwise data bit =1
0056 0935       00226           call    BITOUT          ; serial data out
0057 0370       00227           rlf     txbuf, F        ; rotate txbuf left
0058 02F1       00228           decfsz  count, F        ; 8 bits done?
0059 0A53       00229           goto    TXLP            ; no - go again
005A 0943       00230           call    BITIN           ; read ack bit
                00231           ;
005B 0800       00232           retlw   0
                00233 ;                 ;
                00234 ;*****************************************************************
                00235 ;       Power up routine
                00236 ;       This is the program entry point, which in this case simply
                00237 ;       sets the port_a I/O lines and directs control to the
                00238 ;       write routine.
                00239 ;*****************************************************************
005C            00240 PWRUP
005C 0C00       00241           movlw   b'00000000'
005D 0005       00242           tris    port_a          ; set port A as all output
005E 0065       00243           clrf    port_a          ; all output lines low
005F 0A60       00244           goto    WRBYTE
                00245
                00246 ;*****************************************************************
                00247 ;       Byte Write Routine with data polling technique
                00248 ;
                00249 ;       This routine writes the data in "datao" to
                00250 ;       8 consecutive bytes in the serial EE device starting
                00251 ;       at address 00. To determine when the write cycle time
                00252 ;       of the device, the 'data polling' method is used.  This
```


© 1997 Microchip Technology Inc.

```
                    00253 ;       involves sending a start bit and control byte to the part
                    00254 ;       and checking for an acknowledge.  If the ack bit is low, then
                    00255 ;       the device is through writing, otherwise the the sequence
                    00256 ;       is repeated.  If no low acknowledge is found within 40
                    00257 ;       attempts (about 10 milliseconds) then the routine times
                    00258 ;       out and sets the timeout LED (pin 18) high.  This program
                    00259 ;       will repeat forever.
                    00260 ;
                    00261 ;*****************************************************************
                    00262 ;
0060                00263 WRBYTE
                    00264 ;
0060 0065           00265          clrf    port_a          ; all LEDs off
0061 0CA0           00266          movlw   b'10100000'     ; set slave address and write mode
0062 002F           00267          movwf   slave
0063 0CAA           00268          movlw   b'10101010'     ; set data to write as AAh
0064 002E           00269          movwf   datao
0065 0C08           00270          movlw   .8              ; set number of bytes
0066 0032           00271          movwf   bcount          ; to write as to 8
0067 006C           00272          clrf    addr            ; set starting address to 00
                    00273                                  ;
0068 090E           00274 byte     call    BSTART          ; generate start bit
0069 020F           00275          movf    slave,w         ; move slave address
006A 0030           00276          movwf   txbuf           ; into transmit buffer
006B 0951           00277          call    TX              ; and send it
006C 020C           00278          movf    addr,w          ; move word address
006D 0030           00279          movwf   txbuf           ; into transmit buffer
006E 0951           00280          call    TX              ; and send it
006F 020E           00281          movf    datao,w         ; move data byte
0070 0030           00282          movwf   txbuf           ; to tranmit buffer
0071 0951           00283          call    TX              ; and transmit it
0072 0923           00284          call    BSTOP           ; generate stop bit
                    00285                                  ;
                    00286                                  ; now start polling for a low ack bit
                    00287                                  ;
0073 0C28           00288          movlw   .40             ;
0074 0037           00289          movwf   pollcnt         ; set max number of times to poll as 40
0075 090E           00290 poll     call    BSTART          ; generate start bit
0076 0CA0           00291          movlw   b'10100000'     ; move slave address (write mode)
0077 0030           00292          movwf   txbuf           ; into transmit buffer
0078 0951           00293          call    TX              ; and send it
0079 07EA           00294          btfss   eeprom,di       ; was the ack bit low?
007A 0A7E           00295          goto    exitpoll        ; yes, do another byte
007B 02F7           00296          decfsz  pollcnt, F      ; is poll counter down to zero?
007C 0A75           00297          goto    poll            ; no, poll again. Otherwise the part is
007D 0545           00298          bsf     port_a,timeout  ; not responding in time so set timeout
                    00299                                  ; LED and continue on
                    00300                                  ;
007E 02AC           00301 exitpoll incf    addr, F         ; add 1 to address counter
007F 02F2           00302          decfsz  bcount, F       ; all 8 bytes written?
0080 0A68           00303          goto    byte            ; no, do another byte
0081 0A60           00304          goto    WRBYTE          ; yes, start over
                    00305 ;
                    00306 ;
                    00307      END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XX-------------- ---------------- ---------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   131
Program Memory Words Free:   381
```


© 1997 Microchip Technology Inc.

```
Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX D:2WPAGEW.ASM

```
MPASM 01.40 Released          2WPAGEW.ASM   1-16-1997  16:51:52         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54,c=132
                     00002 ;*****************************************************************
                     00003 ;       2-Wire Page Write Program (122 bytes)
                     00004 ;
                     00005 ;       This program demonstrates how to interface a
                     00006 ;       Microchip PIC 16C54 to a 24LCXX Serial EE device
                     00007 ;       and perform a page write operation on 8 consecutive
                     00008 ;       addresses.
                     00009 ;
                     00010 ;       All of Microchip's 2-wire serial E≤ devices have a
                     00011 ;       buffer or 'page' that can be used as a more efficient
                     00012 ;       method of writing to consecutive addresses.  Some
                     00013 ;       devices have page lengths of 2 bytes, others have
                     00014 ;       page lengths of 4,8,16 or 64 bytes.  Please consult
                     00015 ;       the databook for the page length of the device you
                     00016 ;       are using.  THIS ROUTINE IS WRITTEN FOR A DEVICE
                     00017 ;       WITH A PAGE LENGTH OF 8 OR MORE BYTES.
                     00018 ;
                     00019 ;       When using page mode, the control byte and address
                     00020 ;       are sent for the first address only.  After the data
                     00021 ;       byte for the first address is sent, the data for the
                     00022 ;       next consecutive address is clocked in.  This is
                     00023 ;       repeated as many times as needed (as long as the page
                     00024 ;       length is not exceeded) and then a stop bit is sent.
                     00025 ;       The device will still acknowledge between every byte of
                     00026 ;       data.  After the stop bit is sent, the part will
                     00027 ;       initiate the self timed write cycle.  For all of
                     00028 ;       Microchip's 24LCxx devices, the cycle time for
                     00029 ;       a byte write and a page write is the same.  Therefore,
                     00030 ;       writing 8 bytes in byte mode with a typical 5ms write
                     00031 ;       cycle consumes 40ms of wait time, while the same data
                     00032 ;       written in page mode would consume only 5ms of wait time.
                     00033 ;
                     00034 ;       This routine waits approximately 10mS for the write
                     00035 ;       cycle time, which is enough time for any of the
                     00036 ;       24LCxx devices to complete a write.  A more efficient
                     00037 ;       method of determining when the write cycle is complete
                     00038 ;       is called "data polling."  The data polling method is
                     00039 ;       explained in the program "2wdpoll.asm."  That
                     00040 ;       particular program uses data polling in a byte write
                     00041 ;       mode but it can be used in exactly the same way for
                     00042 ;       a page mode write.
                     00043 ;
                     00044 ;       As an option, the user can connect a LED to pin 18
                     00045 ;       on the PIC 16C54 (use about a 1K resistor in series) as a
                     00046 ;       acknowledge fail indicator.  This LED will come on
                     00047 ;       if the device being programmed does not send a low
                     00048 ;       acknowledge bit at the proper times.  This can be
                     00049 ;       tested by simply running the program with no part
                     00050 ;       in the socket.
                     00051 ;
                     00052 ;       Timing is based on using the PIC 16C5x in 'XT' mode
                     00053 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00054 ;       will be approximately 40 Khz for this setup.
```


© 1997 Microchip Technology Inc.

```
                        00055 ;
                        00056 ;
                        00057 ;         PIC 16C54 to Serial EE Connections:
                        00058 ;
                        00059 ;           PIC 16C54       Serial EE
                        00060 ;         ------------    ------------
                        00061 ;         Pin 12 (RB6) -->  SCLK
                        00062 ;         Pin 13 (RB7) -->  SDATA
                        00063 ;
                        00064 ;         PIN 18 (RA1) -->  Acknowledge fail LED  (Optional)
                        00065 ;
                        00066 ;
                        00067 ;
                        00068 ;         Program:          2WPAGEW.ASM
                        00069 ;         Revision Date:
                        00070 ;                           1-15-97      Compatibility with MPASMWIN 1.40
                        00071 ;
                        00072 ;*****************************************************************
                        00073 ;             Register Definitions
                        00074 ;*****************************************************************
  00000005              00075 port_a  equ     5h      ; port 5 (port a) used for LED outputs
  00000006              00076 port_b  equ     6h      ; port 6 (port b) used for data and
                        00077                         ; clock lines
  0000000A              00078 eeprom  equ     0ah     ; bit buffer
  0000000C              00079 addr    equ     0ch     ; address counter
  0000000E              00080 datao   equ     0eh     ; data output register
  0000000F              00081 slave   equ     0fh     ; device address (1010xxx0)
  00000010              00082 txbuf   equ     10h     ; transmit buffer
  00000011              00083 count   equ     11h     ; bit counter
  00000012              00084 bcount  equ     12h     ; byte counter
  00000015              00085 loops   equ     15h     ; delay loop counter
  00000016              00086 loops2  equ     16h     ; delay loop counter
                        00087 ;*****************************************************************
                        00088 ;             Bit Definitions
                        00089 ;*****************************************************************
  00000007              00090 di      equ     7       ; eeprom input bit
  00000006              00091 do      equ     6       ; eeprom output bit
  00000007              00092 sdata   equ     7       ; serial EE data line (port_b,pin 13)
  00000006              00093 sclk    equ     6       ; serail EE clock line (port_b,pin 12)
  00000001              00094 ackf    equ     1       ; acknowledge fail LED, port_a (pin 18)
                        00095 ;*****************************************************************
                        00096 ;
                        00097 ;
01FF                    00098         org     01ffh   ; set reset vector
01FF 0A5E               00099         goto    PWRUP
0000                    00100         org     000h    ;
0000 0A5E               00101         goto    PWRUP
                        00102 ;
                        00103 ;*****************************************************************
                        00104 ;       DELAY ROUTINE
                        00105 ;       This routine takes the value in 'loops'
                        00106 ;       and multiplies it times 1 millisecond to
                        00107 ;       determine delay time.
                        00108 ;*****************************************************************
0001                    00109 WAIT
                        00110 ;
0001 0C6E               00111 top      movlw   .110           ; timing adjustment variable
0002 0036               00112          movwf   loops2
0003 0000               00113 top2     nop                    ; sit and wait
0004 0000               00114          nop
0005 0000               00115          nop
0006 0000               00116          nop
0007 0000               00117          nop
0008 0000               00118          nop
0009 02F6               00119          decfsz  loops2, F      ; inner loops complete?
000A 0A03               00120          goto    top2           ; no, go again
```


© 1997 Microchip Technology Inc.

```
                  00121                                 ;
000B 02F5         00122          decfsz  loops, F       ; outer loops complete?
000C 0A01         00123          goto    top            ; no, go again
000D 0800         00124          retlw   0              ; yes, return from sub
                  00125 ;
                  00126 ;*****************************************************************
                  00127 ;       Start Bit Subroutine
                  00128 ;       this routine generates a start bit
                  00129 ;       (Low going data line while clock is high)
                  00130 ;*****************************************************************
                  00131 ;
000E              00132 BSTART
000E 05E6         00133          bsf     port_b,sdata   ; make sure data is high
000F 0C3F         00134          movlw   b'00111111'
0010 0006         00135          tris    port_b         ; set data and clock lines for output
0011 04C6         00136          bcf     port_b,sclk    ; make sure clock is low
0012 0000         00137          nop
0013 05C6         00138          bsf     port_b,sclk    ; set clock high
0014 0000         00139          nop
0015 0000         00140          nop
0016 0000         00141          nop
0017 0000         00142          nop
0018 0000         00143          nop
0019 04E6         00144          bcf     port_b,sdata   ; data line goes low during
                  00145                                 ; high clock for start bit
001A 0000         00146          nop
001B 0000         00147          nop
001C 0000         00148          nop
001D 0000         00149          nop
001E 0000         00150          nop                    ; timing adjustment
001F 04C6         00151          bcf     port_b,sclk    ; start clock train
0020 0000         00152          nop
0021 0000         00153          nop
0022 0800         00154          retlw   0
                  00155          ;
                  00156          ;    End of Subroutine
                  00157 ;*****************************************************************
                  00158 ;       Stop Bit Subroutine
                  00159 ;       This routine generates a stop bit
                  00160 ;       (High going data line while clock is high)
                  00161 ;*****************************************************************
0023              00162 BSTOP
0023 0C3F         00163          movlw   b'00111111'    ;
0024 0006         00164          tris    port_b         ; set data/clock lines as outputs
0025 04E6         00165          bcf     port_b,sdata   ; make sure data line is low
0026 0000         00166          nop
0027 0000         00167          nop
0028 0000         00168          nop
0029 05C6         00169          bsf     port_b,sclk    ; set clock high
002A 0000         00170          nop
002B 0000         00171          nop
002C 0000         00172          nop
002D 05E6         00173          bsf     port_b,sdata   ; data goes high while clock high
                  00174                                 ; for stop bit
002E 0000         00175          nop
002F 0000         00176          nop
0030 04C6         00177          bcf     port_b,sclk    ; set clock low again
0031 0000         00178          nop
0032 0000         00179          nop
0033 0000         00180          nop
0034 0800         00181          retlw   0
                  00182 ;
                  00183 ;           End of Subroutine
                  00184 ;*****************************************************************
                  00185 ;       BITOUT routine
                  00186 ;       This routine takes one bit of data in 'do' and
```


© 1997 Microchip Technology Inc.

```
                    00187 ;       transmits it to the serial EE device
                    00188 ;*************************************************************
0035                00189 BITOUT
0035 0C3F           00190         movlw   b'00111111'     ; set data,clock as outputs
0036 0006           00191         tris    port_b
0037 07CA           00192         btfss   eeprom,do       ; check for state of data bit to xmit
0038 0A3B           00193         goto    bitlow          ;
0039 05E6           00194         bsf     port_b,sdata    ; set data line high
003A 0A3C           00195         goto    clkout          ; go toggle the clock
                    00196
003B 04E6           00197 bitlow  bcf     port_b,sdata    ; output a low bit
003C 05C6           00198 clkout  bsf     port_b,sclk     ; set clock line high
003D 0000           00199         nop
003E 0000           00200         nop
003F 0000           00201         nop
0040 0000           00202         nop
0041 04C6           00203         bcf     port_b,sclk     ; return clock line low
0042 0800           00204         retlw   0
                    00205 ;
                    00206 ;                  End of Subroutine
                    00207 ;
                    00208 ;*************************************************************
                    00209 ;       BITIN routine reads one bit of data from the
                    00210 ;       serial EE device and stores it in 'di'
                    00211 ;*************************************************************
0043                00212 BITIN
0043 05EA           00213         bsf     eeprom,di       ; assume input bit is high
0044 0CBF           00214         movlw   b'10111111'     ; make sdata an input line
0045 0006           00215         tris    port_b
0046 05E6           00216         bsf     port_b,sdata    ; set sdata line for input
0047 05C6           00217         bsf     port_b,sclk     ; set clock line high
0048 0000           00218         nop                     ; just sit here a sec
0049 0000           00219         nop
004A 0000           00220         nop
004B 0000           00221         nop
004C 0000           00222         nop                     ;
004D 07E6           00223         btfss   port_b,sdata    ; read the data bit
004E 04EA           00224         bcf     eeprom,di       ; input bit was low
004F 04C6           00225         bcf     port_b,sclk     ; set clock line low
                    00226                                 ;
0050 0800           00227         retlw   0               ;
                    00228 ;
                    00229 ;*****************************************************************
                    00230 ;       Transmit Data Subroutine
                    00231 ;       This routine takes the byte of data stored in the
                    00232 ;       'datao' register and transmits it to the serial EE device.
                    00233 ;       It will then send 1 more clock to the serial EE for the
                    00234 ;       acknowledge bit.  If the ack bit from the part was low
                    00235 ;       then the transmission was sucessful.  If it is high, then
                    00236 ;       the device did not send a proper ack bit and the ack
                    00237 ;       fail LED will be turned on.
                    00238 ;*****************************************************************
0051                00239 TX
0051 0C08           00240          movlw   .8
0052 0031           00241          movwf   count          ; set the #bits to 8
                    00242          ;
0053                00243 TXLP
0053 04CA           00244          bcf     eeprom,do      ; assume bit out is low
0054 06F0           00245          btfsc   txbuf,7        ; is bit out really low?
0055 05CA           00246          bsf     eeprom,do      ; otherwise data bit =1
0056 0935           00247          call    BITOUT         ; serial data out
0057 0370           00248          rlf     txbuf, F       ; rotate txbuf left
0058 02F1           00249          decfsz  count, F       ; 8 bits done?
0059 0A53           00250          goto    TXLP           ; no - go again
005A 0943           00251          call    BITIN          ; read ack bit
005B 06EA           00252          btfsc   eeprom,di      ; check ack bit
```


© 1997 Microchip Technology Inc.

```
005C 0525           00253          bsf     port_a,ackf     ; set acknowledge fail LED if the
                    00254                                  ; device did not pull data low
                    00255          ;
005D 0800           00256          retlw   0
                    00257 ;
                    00258 ;*****************************************************************
                    00259 ;       Power up routine
                    00260 ;       This is the program entry point, which in this case simply
                    00261 ;       sets the port_a I/O lines and directs control to the Page
                    00262 ;       write routine.
                    00263 ;*****************************************************************
005E                00264 PWRUP
005E 0C00           00265           movlw  b'00000000'
005F 0005           00266           tris   port_a          ; set port A as all output
0060 0065           00267           clrf   port_a          ; all output lines low
0061 0A62           00268           goto   WRPAGE
                    00269
                    00270 ;*****************************************************************
                    00271 ;       Page Write Routine
                    00272 ;
                    00273 ;       This routine uses page mode to write the data in "datao" to
                    00274 ;       8 consecutive bytes in the serial EE device starting
                    00275 ;       at address 00. This routine waits 10mS after every
                    00276 ;       page to give the device time to do the write.  This
                    00277 ;       routine executes forever
                    00278 ;*****************************************************************
                    00279 ;
0062                00280 WRPAGE
                    00281 ;
0062 0065           00282          clrf    port_a          ; clear all LEDs
0063 0CA0           00283          movlw   b'10100000'     ; set slave address and write mode
0064 002F           00284          movwf   slave
0065 0C55           00285          movlw   b'01010101'     ; set data to write as 55h
0066 002E           00286          movwf   datao
                    00287                                  ;
0067 0C08           00288          movlw   .8              ; set number of bytes
0068 0032           00289          movwf   bcount          ; to write as to 8
0069 006C           00290          clrf    addr            ; set starting address to 00
                    00291                                  ;
006A 090E           00292          call    BSTART          ; generate start bit
006B 020F           00293          movf    slave,w         ; move slave address
006C 0030           00294          movwf   txbuf           ; into transmit buffer
006D 0951           00295          call    TX              ; and send it
                    00296
006E 020C           00297          movf    addr,w          ; move word address
006F 0030           00298          movwf   txbuf           ; into transmit buffer
0070 0951           00299          call    TX              ; and send it
                    00300
0071 020E           00301 byte     movf    datao,w         ; move data byte
0072 0030           00302          movwf   txbuf           ; to tranmit buffer
0073 0951           00303          call    TX              ; and transmit it
0074 02F2           00304          decfsz  bcount, F       ; all 8 bytes written?
0075 0A71           00305          goto    byte            ; no, do another
0076 0923           00306          call    BSTOP           ; yes,generate stop bit
                    00307                                  ;
0077 0C0A           00308          movlw   .10
0078 0035           00309          movwf   loops           ; set delay time to give
0079 0901           00310          call    WAIT            ; 10 ms wait after every byte
                    00311                                  ;
007A 0A62           00312          goto    WRPAGE          ; start over
                    00313 ;
                    00314 ;
                    00315      END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
```


© 1997 Microchip Technology Inc.

```
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX-----
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   124
Program Memory Words Free:   388


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX E:2WSEQR.ASM

```
MPASM 01.40 Released           2WSEQR.ASM   1-16-1997  16:52:18         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001          LIST P=16C54,c=132
                     00002 ;*****************************************************************
                     00003 ;       2-Wire Sequential Read Program (120 bytes)
                     00004 ;
                     00005 ;       This program demonstrates how to interface a
                     00006 ;       Microchip PIC 16C54 to a 24LCXX Serial EE device
                     00007 ;       and perform a sequential read operation.  A sequential
                     00008 ;       read involves setting the address pointer once and then
                     00009 ;       using the auto-increment ability of the part to read
                     00010 ;       consecutive addresses by simply providing more clocks.
                     00011 ;
                     00012 ;       Timing is based on using the PIC 16C54 in 'XT' mode
                     00013 ;       using a 4Mhz crystal.  Clock speeds to the serial EE
                     00014 ;       will be approximately 60 Khz for this setup.
                     00015 ;
                     00016 ;       As an option, the user may connect a LED to pin 18
                     00017 ;       on the PIC 16C54 (use about a 1K resistor in series) as an
                     00018 ;       acknowledge fail indicator.  This LED will come on if
                     00019 ;       the serial EE fails to acknowledge correctly.
                     00020 ;
                     00021 ;        PIC 16C54 to Serial EE Connections:
                     00022 ;
                     00023 ;          PIC 16C54        Serial EE
                     00024 ;       ------------     ------------
                     00025 ;       Pin 12 (RB6) -->  SCLK
                     00026 ;       Pin 13 (RB7) -->  SDATA
                     00027 ;
                     00028 ;       PIN 18 (RA1) -->  Acknowledge fail LED  (Optional)
                     00029 ;
                     00030 ;
                     00031 ;       Program:          2WSEQR.ASM
                     00032 ;       Revision Date:
                     00033 ;                         1-15-97      Compatibility with MPASMWIN 1.40
                     00034 ;
                     00035 ;*****************************************************************
                     00036 ;           Register Definitions
                     00037 ;*****************************************************************
00000003             00038 status  equ     3h      ; status register
00000005             00039 port_a  equ     5h      ; port 5 (port_a) used for LEDs
00000006             00040 port_b  equ     6h      ; port 6 (port b) used for data and
                     00041                         ; clock lines
0000000A             00042 eeprom  equ    0ah      ; bit buffer
0000000B             00043 bycnt   equ    0bh      ; byte counter for read mode
0000000C             00044 addr    equ    0ch      ; address counter
0000000D             00045 datai   equ    0dh      ; data input register
0000000E             00046 datao   equ    0eh      ; data output register
0000000F             00047 slave   equ    0fh      ; device address (1010xxx0)
00000010             00048 txbuf   equ    10h      ; transmit buffer
00000011             00049 count   equ    11h      ; bit counter
00000012             00050 bcount  equ    12h      ; byte counter
00000015             00051 loops   equ    15h      ; delay loop counter
00000016             00052 loops2  equ    16h      ; delay loop counter
                     00053 ;*****************************************************************
                     00054 ;           Bit Definitions
```


© 1997 Microchip Technology Inc.

```
                    00055 ;*************************************************************
  00000007          00056 di      equ    7       ; eeprom input bit
  00000006          00057 do      equ    6       ; eeprom output bit
  00000007          00058 sdata   equ    7       ; serial EE data line (port_b,pin 13)
  00000006          00059 sclk    equ    6       ; serail EE clock line (port_b,pin 12)
  00000001          00060 ackf    equ    1       ; acknowledge fail LED, port_a
                    00061 ;*************************************************************
                    00062 ;
                    00063 ;
01FF                00064         org    01ffh   ; set reset vector
01FF 0A5B           00065         goto   PWRUP
0000                00066         org    000h    ;
0000 0A5B           00067         goto   PWRUP
                    00068 ;
                    00069 ;*************************************************************
                    00070 ;       Start Bit Subroutine
                    00071 ;       this routine generates a start bit
                    00072 ;       (Low going data line while clock is high)
                    00073 ;*************************************************************
                    00074 ;
0001                00075 BSTART
0001 05E6           00076          bsf     port_b,sdata   ; make sure data is high
0002 0C3F           00077          movlw   b'00111111'
0003 0006           00078          tris    port_b         ; set data and clock lines for output
0004 04C6           00079          bcf     port_b,sclk    ; make sure clock is low
0005 0000           00080          nop
0006 05C6           00081          bsf     port_b,sclk    ; set clock high
0007 0000           00082          nop
0008 0000           00083          nop
0009 0000           00084          nop
000A 0000           00085          nop
000B 0000           00086          nop
000C 04E6           00087          bcf     port_b,sdata   ; data line goes low during
                    00088                                 ; high clock for start bit
000D 0000           00089          nop
000E 0000           00090          nop
000F 0000           00091          nop
0010 0000           00092          nop
0011 0000           00093          nop                    ; timing adjustment
0012 04C6           00094          bcf     port_b,sclk    ; start clock train
0013 0000           00095          nop
0014 0000           00096          nop
0015 0800           00097          retlw   0
                    00098          ;
                    00099          ;    End of Subroutine
                    00100 ;*************************************************************
                    00101 ;       Stop Bit Subroutine
                    00102 ;       This routine generates a stop bit
                    00103 ;       (High going data line while clock is high)
                    00104 ;*************************************************************
0016                00105 BSTOP
0016 0C3F           00106          movlw   b'00111111'    ;
0017 0006           00107          tris    port_b         ; set data/clock lines as outputs
0018 04E6           00108          bcf     port_b,sdata   ; make sure data line is low
0019 0000           00109          nop
001A 0000           00110          nop
001B 0000           00111          nop
001C 05C6           00112          bsf     port_b,sclk    ; set clock high
001D 0000           00113          nop
001E 0000           00114          nop
001F 0000           00115          nop
0020 05E6           00116          bsf     port_b,sdata   ; data goes high while clock high
                    00117                                 ; for stop bit
0021 0000           00118          nop
0022 0000           00119          nop
0023 04C6           00120          bcf     port_b,sclk    ; set clock low again
```

7

© 1997 Microchip Technology Inc.

```
0024 0000          00121          nop
0025 0000          00122          nop
0026 0000          00123          nop
0027 0800          00124          retlw   0
                   00125 ;
                   00126 ;           End of Subroutine
                   00127 ;*************************************************************
                   00128 ;       BITOUT routine takes one bit of data in 'do' and
                   00129 ;       transmits it to the serial EE device
                   00130 ;*************************************************************
0028               00131 BITOUT
0028 0C3F          00132          movlw   b'00111111'    ; set data,clock as outputs
0029 0006          00133          tris    port_b
002A 07CA          00134          btfss   eeprom,do      ; check for state of data bit to xmit
002B 0A2E          00135          goto    bitlow         ;
002C 05E6          00136          bsf     port_b,sdata   ; set data line high
002D 0A2F          00137          goto    clkout         ; go toggle the clock
                   00138
002E 04E6          00139 bitlow   bcf     port_b,sdata   ; output a low bit
002F 05C6          00140 clkout   bsf     port_b,sclk    ; set clock line high
0030 0000          00141          nop
0031 0000          00142          nop
0032 0000          00143          nop
0033 0000          00144          nop
0034 04C6          00145          bcf     port_b,sclk    ; return clock line low
0035 0800          00146          retlw   0
                   00147 ;
                   00148 ;                  End of Subroutine
                   00149 ;
                   00150 ;*************************************************************
                   00151 ;       BITIN routine reads one bit of data from the
                   00152 ;       serial EE device and stores it in 'di'
                   00153 ;*************************************************************
0036               00154 BITIN
0036 05EA          00155          bsf     eeprom,di      ; assume input bit is high
0037 0CBF          00156          movlw   b'10111111'    ; make sdata an input line
0038 0006          00157          tris    port_b
0039 05C6          00158          bsf     port_b,sclk    ; set clock line high
003A 0000          00159          nop                    ; just sit here a sec
003B 0000          00160          nop
003C 0000          00161          nop
003D 0000          00162          nop
003E 0000          00163          nop                    ;
003F 07E6          00164          btfss   port_b,sdata   ; read the data bit
0040 04EA          00165          bcf     eeprom,di      ; input bit was low, set 'di' accordingly
0041 04C6          00166          bcf     port_b,sclk    ; set clock line low
                   00167                                 ;
0042 0800          00168          retlw   0
                   00169 ;
                   00170 ;*************************************************************
                   00171 ;       Transmit Data Subroutine
                   00172 ;       This routine takes the byte of data stored in the
                   00173 ;       'datao' register and transmits it to the serial EE device.
                   00174 ;       It will then send 1 more clock to the serial EE for the
                   00175 ;       acknowledge bit.  If the ack bit from the part was low
                   00176 ;       then the transmission was sucessful.  If it is high, then
                   00177 ;       the device did not send a proper ack bit and the ack
                   00178 ;       fail LED will be turned on.
                   00179 ;*************************************************************
0043               00180 TX
0043 0C08          00181          movlw   .8
0044 0031          00182          movwf   count          ; set the #bits to 8
                   00183          ;
0045               00184 TXLP
0045 04CA          00185          bcf     eeprom,do      ; assume bit out is low
0046 06F0          00186          btfsc   txbuf,7        ; is bit out really low?
```


© 1997 Microchip Technology Inc.

```
0047 05CA  00187          bsf     eeprom,do       ; no, set it high
0048 0928  00188          call    BITOUT          ; send the bit to serial EE
0049 0370  00189          rlf     txbuf, F        ; rotate txbuf left
004A 02F1  00190          decfsz  count, F        ; 8 bits done?
004B 0A45  00191          goto    TXLP            ; no - go again
004C 0936  00192          call    BITIN           ; read ack bit
004D 06EA  00193          btfsc   eeprom,di       ; check ack bit
004E 0525  00194          bsf     port_a,ackf     ; set acknowledge fail LED if the
           00195                                  ; device did not pull data low
           00196          ;
004F 0800  00197          retlw   0
           00198 ;
           00199 ;*****************************************************************
           00200 ;       Receive data Routine
           00201 ;       This routine reads one byte of data from the part
           00202 ;       into the 'datai' register.  It then sends a high
           00203 ;       ack bit to indicate that no more data is to be read
           00204 ;*****************************************************************
0050       00205 RX
0050 0C08  00206          movlw   .8              ; set # bits to 8
0051 0031  00207          movwf   count
0052 006D  00208          clrf    datai           ; clear input register
0053 0403  00209          bcf     status,0        ; make sure carry bit is low
0054 036D  00210 RXLP     rlf     datai, F        ; rotate datai 1 bit left
0055 0936  00211          call    BITIN           ; read a bit
0056 06EA  00212          btfsc   eeprom,di
0057 050D  00213          bsf     datai,0         ; set bit 0 if necessary
0058 02F1  00214          decfsz  count, F        ; 8 bits done?
0059 0A54  00215          goto    RXLP            ; no, do another
005A 0800  00216          retlw   0
           00217 ;
           00218 ;*****************************************************************
           00219 ;       Power up routine
           00220 ;       This is the program entry point, which in this case simply
           00221 ;       sets the port_a I/O lines and directs control to the
           00222 ;       Read routine.
           00223 ;*****************************************************************
005B       00224 PWRUP
005B 0C00  00225            movlw  b'00000000'
005C 0005  00226            tris   port_a           ; set port A as all output
005D 0065  00227            clrf   port_a           ; all output lines low
005E 0A5F  00228            goto   READ
           00229
           00230 ;
           00231 ;*************************************************************
           00232 ;       READ (sequential read routine)
           00233 ;
           00234 ;       This routine reads 8 consecutive addresses of the
           00235 ;       serial EE device starting at address 00 in the
           00236 ;       sequential read mode. Reading in this mode is more
           00237 ;       efficient than the random read mode as the control byte
           00238 ;       and address have to be sent only once at the beginning
           00239 ;       of the sequence.  As many consecutive addresses as
           00240 ;       needed can then be read from the part until a stop bit is
           00241 ;       sent. In the read mode, the PIC16C54 must send the acknowledge
           00242 ;       bit after every 8 data bits from the device.  When the
           00243 ;       last byte needed has been read, then the controller will
           00244 ;       send a high acknowledge bit and then a stop bit to halt
           00245 ;       transmission from the device.
           00246 ;*************************************************************
005F       00247 READ
           00248                                  ;
005F 0425  00249            bcf     port_a,ackf ; clear the ack fail LED if on
0060 0C08  00250            movlw   .8
0061 0032  00251            movwf   bcount      ; set number of bytes to read as 8
0062 0CA0  00252            movlw   b'10100000' ; set slave address and write mode
```


© 1997 Microchip Technology Inc.

```
0063 002F           00253          movwf   slave
0064 006C           00254          clrf    addr         ; set starting address to 00
                    00255                               ;
0065 0901           00256          call    BSTART       ; generate start bit
0066 020F           00257          movf    slave,w      ; get slave address
0067 0030           00258          movwf   txbuf        ; into transmit buffer
0068 0943           00259          call    TX           ; and send it
0069 020C           00260          movf    addr,w       ; get word address
006A 0030           00261          movwf   txbuf        ; into transmit buffer
006B 0943           00262          call    TX           ; and send it
006C 0901           00263          call    BSTART       ; generate start bit
006D 0CA1           00264          movlw   b'10100001'  ; get slave address and read mode
006E 0030           00265          movwf   txbuf        ; into transmit buffer
006F 0943           00266          call    TX           ; and transmit it
                    00267                               ;
0070 0950           00268 rbyte    call    RX           ; read 1 byte from device
0071 02F2           00269          decfsz  bcount, F    ; are all 8 bytes read?
0072 0A77           00270          goto    lowack       ; no, send low ack and do another
0073 05CA           00271          bsf     eeprom,do    ; yes, send high ack bit
0074 0928           00272          call    BITOUT       ; to stop tranxmission
0075 0916           00273          call    BSTOP        ; and send a stop bit
0076 0A5F           00274          goto    READ         ; start all over
                    00275
0077 04CA           00276 lowack   bcf     eeprom,do    ; send low ack bit
0078 0928           00277          call    BITOUT       ; to continue transmission
0079 0A70           00278          goto    rbyte        ; and read another byte
                    00279 ;
                    00280      END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXX------
01C0 : ---------------- ---------------- ---------------- --------------X

All other memory blocks unused.

Program Memory Words Used:   123
Program Memory Words Free:   389


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN572

Questions and Answers Concerning Serial EEPROMs

Authors: *Bruce Negley*
Microchip Technology Inc.

Interfacing a serial EEPROM to a controller can be a confusing task, especially for first time users. Because communication with a parallel memory device is more straightforward, the advantages of serial devices are sometimes overlooked because of the time required to learn a different protocol. In order to minimize this time, Microchip provides many application notes on how to interface a controller to our serial memory products. Since every application is different, questions on different aspects of serial memories still arise. This application note provides answers to some of these commonly asked questions on using serial EEPROMs. This series of questions and answers covers a broad range of topics concerning both 2-wire and 3-wire serial EEPROMS, and covers material and/or circumstances that may not be provided in the data book.

QUESTIONS ON 2-WIRE DEVICES (24XXXX OR 85XXXX)

Q: What happens if a STOP bit is inadvertently sent to the device in the middle of a page write command?

A: It depends on which device you are using. The older 24Cxx devices will write any fully loaded bytes in the page to the device. For example if the stop bit occurs before all bits are loaded in the 5th byte, then the first 4 bytes will be written and the 5th byte will not. On the 24LCxxB or 24AAxx devices, if the stop bit occurs before all 8 bits in a byte are loaded, then the entire command will abort and no bytes will be written.

Q: What happens if a START bit is inadvertently sent to the device in the middle of command?

A: A start bit will always reset the part. It should be noted that if the device has control of the SDA line for an acknowledge bit or a read command, it may not be possible to send a start bit.

Q: I'm using interrupts in my controller code, and it is possible that I may jump out of my serial EEPROM communication routine to handle an interrupt. Is there any problem with leaving the device sitting in the middle of a command as long as the clock remains stable?

A: It is perfectly valid to leave a command in the middle indefinitely as long as the clock line remains stable.

Q: I'm using an interrupt routine in my controller code that may leave the serial EEPROM in the middle of a read command. When I return from the interrupt, is there any way to reset the part to make sure it is in a known state before I start sending it commands again?

A: If the part was in the process of outputting data, you can always get it back to standby mode by allowing the SDA line to float high and give it 9 clocks. This assures that the device will not receive the acknowledge bit at the end of the current byte and will abort the command and go to standby.

Q: I'm using a 24LCxxB in my application. What happens if the WP line goes from low (unprotected) to high (protected) in the middle of a write command?

A: The state of the WP pin is OR'd with the write control circuitry inside the device. The state of the pin is not latched at any point of the command to see if the write should be done or not. Therefore, if the WP pin went high before the stop bit was sent to initiate the command, then the array would be protected. If the WP pin went high after the stop bit was sent, then the outcome is unpredictable, because the result would vary according to how far into command the WP pin went high.

Q: I'm attempting to use a serial EEPROM in an application for the first time, and am having problems getting it to work. I have followed all the timing diagrams in the databook but when I attempt to read data from the device I only read zeros.

Q: When a read command is initiated and all the data is zeros, the problem is most likely one of two things:

1. A pull-up resistor on the SDA line is required for the device to operate correctly. Common values are 10K - 20K ohms. If this pull-up is not provided, then the device can only output zeros.
2. The controller is not releasing the bus to the device and is holding it low. Check the controller code and make sure that the SDA line has been set for input during the read command.

7

© 1997 Microchip Technology Inc.

Q: I had been using a 24C04A in my application and have switched to the 24LC04B so I can run at a lower voltage. Now the circuit is no longer working correctly. The read seems to work fine but the write seems to not work at all or only intermittently.

A: A: The problem is probably with the difference in write cycle times between the two devices. The write cycle time for the 24C04A is about 0.5ms per byte, while the cycle time for the 24LC04B is around 2-4ms per byte. Therefore, when you send consecutive write commands using the new device, it is ignoring all commands that occur while it is in the middle of a write cycle.

Q: I'm using a 24LC16B device in my application. Is there any way I can determine when the write cycle is complete instead of waiting the maximum 10ms?

A: Yes, you can determine when the write cycle is complete on any of our 2-wire devices by utilizing a simple technique called "data polling" or "acknowledge polling." Data polling is done after the stop bit has been sent to the device to initiate a write cycle. You simply send the device a start bit followed by a proper control byte with the R/W bit set low. You then check the status of the acknowledge bit. If the acknowledge bit is high, then the device is still busy writing. You can continue to send the start bit and control byte until the device acknowledges. When the acknowledge bit reads low, then the device has finished the write cycle. You can now send the address and data bytes if you are going to do a write command, or just send the address if you are going to do a read command. Of course you may also send a start bit and begin the next command from scratch.

Q: I'm confused about when the acknowledge bit from the device actually occurs; can you clarify this?

A: The device will attempt to take over the SDA line for the acknowledge bit on the falling edge of the clock for the last bit in the byte. It will then release the SDA line on the falling edge of the clock given for the acknowledge bit. For example, when the control byte is sent to the device, it will take the SDA line low on the falling edge of the clock for the R/W bit. It will then release the SDA line on the next falling edge of the clock (the acknowledge bit). The same sequence occurs for sending the device a byte of data or address.

Q: I am using a 24LC01B in my application which has an 8 byte page buffer. What happens if I load more than 8 bytes into the device before giving it the stop bit?

A: For all 24LCxxB products, the 24C04A and the 85C92, the page buffer will wrap around to the beginning of the buffer and begin to overwrite the data that has previously been loaded For the 24C01A/02A and 85C72/82, the write command will abort if more bytes than the page buffer will hold are loaded.

Q: Can I send a stop bit and a start bit with the same clock pulse?

A: Yes, this is valid as long as setup and hold times for both the start and stop bits are obeyed.

Q: What happens if I give the device a start bit and then a stop bit in the same clock pulse?

A: The 24Cxx devices will ignore the stop bit, the 24LCxxB products will accept the stop bit and go to standby mode.

Q: Does the address pointer get incremented after a current address read command?

A: A current address read command can be halted by sending a stop bit or by sending a high acknowledge bit after the data has been sent by the device. On the 24Cxx devices, ending the read command with a stop bit will increment the address pointer to the next address. Ending the command with a high ack bit will not increment the address.

For the 24LCxxB products, ending the command with either the stop bit or the high ack bit will increment the address pointer.

Q: I am using the 24LC16B and want to write to the part using the page write mode. Can I start a page write at any location in the device?

A: You can start a page write at any location in the device, but you need to be aware that the part is arranged in 16-byte pages and you cannot cross a page boundary. If you attempt a page write that goes beyond the end of the current page, data loaded will 'roll around' to the beginning of the current page instead of going to the next one. For example; the first page in the device starts at address 0x00 and ends at 0x0F. If you start a page write at address 0x0E, the first byte loaded will be written at address 0x0E, the second at 0x0F and the third byte loaded will roll around to the beginning of the page and be written to address 0x00.

© 1997 Microchip Technology Inc.

QUESTIONS AND ANSWERS FOR 3-WIRE DEVICES (93XXXX)

Q: What would happen if during a write command I inadvertently sent either too many or too few clocks to the device before dropping the CS line?

A: If you send too many clocks to the device and then drop the CS line to initiate the write cycle, the extra clocks will be ignored but the command will execute. If you do not send enough clocks and then drop the CS line, then the command will abort and no write will take place.

Q: I am using a 93LC56 device in my application and am having problems getting it to work correctly. The read sequence seems to work fine but I unable to write any data to the part.

A: A problem such as this is usually caused by either not giving the part the required number of bits for the command before dropping the CS line, or not dropping the CS line at all. The write command will not commence until the CS line is brought low.

Q: I am currently using a 93C46 device and am looking to go to the 93LC46 device so I can run at a lower voltage. What are the main differences between these devices?

Q: There are several operational differences between these devices that you should be aware of:

1. The 93LC46 operates in both x8 and x16 modes; the 94C46 is x16 only.
2. The 93LC46 supports the sequential read function the 93C46 does not.
3. The write cycle on the 94LC46 begins on the falling edge of the CS line; the write cycle for tne 93C46 begins on the rising edge of the last clock.

Q: Is the 93LC46 drop-in compatible with the 93C46 or would I be better off using the 93LC46B?

A: The 93LC46 and 93LC46B are the same device with one exception; the ORG pin on the 93LC46B is internally tied so it will only operate in the x16 mode. This allows the user to leave pin 7 floating, just like you would on the 93C46. For this reason, the 93LC46B is a closer match than the 93LC46. Please note the other operational differences described in the previous question.

Q: I am confused as to whether I have to toggle the CS line low in-between every command.

A: Yes, the CS line must go low for at least 250 ns between each command. If you are doing a write command and you bring CS low to activate the data polling mode, you must toggle CS low again after the ready signal has been given by the device before the next start bit can be sent.

© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN601

EEPROM Endurance Tutorial

Authors: Bruce Negley
Microchip Technology Inc.

INTRODUCTION

The endurance of an EEPROM-based device will be quoted by a manufacturer in terms of the minimum number of erase/write cycles (write cycles) that the device is capable of sustaining before failure. A write cycle is generally considered to be the operation that changes data in a device from one value to the next.

There are several EEPROM-based devices available on the market. Microchip Technology Incorporated makes three general types of EEPROM-based product: Serial EEPROMs, Parallel EEPROMs, and EEPROM-based Microcontrollers. As a manufacturer of many EEPROM products, Microchip is concerned with endurance and continues to try to educate its customers on the importance of this topic.

There are many differences in the interpretation of "endurance" that can result in misleading or inaccurate information being used in design decisions. This paper hopes to clear up any questions that the customer may have in the subject of endurance, without becoming so technical that the information given is not helpful.

There is no widely used standard for any type of endurance test. Each manufacturer will use their own endurance testing methodology. This report will describe all the testing options, and which tests Microchip performs on its EEPROM-based products.

The MIL-STD cycling test (Method 1033) has not been updated since 1977 and is well out of date as applied to EEPROM non-volatile memories. The standard does not distinguish the difference between block cycling and byte cycling, and gives a very poor failure criteria. Microchip does not use this standard.

BASIC TERMS

The definition of "endurance" (as applied to EEPROMs) in the first part of this introduction contains various words and phrases that require clear definition and understanding. As shown in the following paragraphs, different manufacturers will use different standards.

"Endurance cycling" is a test performed by all manufacturers (and some customers) to determine how many "write cycles" the product will achieve before failing.

The "minimum number of write cycles" is the least number of times that you can expect to subject the product to a "write cycle" before it fails.

"Failure" is a somewhat arbitrary definition, since a device only truly fails when it no longer meets the customer expectation, and does not operate in his system. A failure can be defined in this, the loosest of definitions, or the most stringent of definitions (whereby a device would fail if it did not meet any of the data sheet parameters), as well as a wide range in between.

For example, if the device did not correctly store data into a particular address that the customer was not using, then the device would work correctly for the customer but would fail a functional test set by the manufacturer. Likewise, if the device drew more current than the data sheet specified after some time, but the customer application could supply the current needed, the device would work in the customer application but would fail a parametric test set by the manufacturer.

Microchip uses the most stringent definition: **A failure occurs when the device fails to meet any data sheet condition under any guaranteed operating condition of temperature and voltage.**

The number of devices that can fail before a particular endurance criteria is not met is also somewhat flexible. Even the most quality conscious manufacturer will occasionally have a failure, so a failure level is defined. The industry standard conditions for many types of reliability tests are set by JEDEC (the Joint Electronic Device Engineering Council). JEDEC defines that if 5% or less of a given sample fails at a given endurance goal, then that goal has been met. For example, if a sample of 100 units are endurance cycled to 1 million cycles and 3 have failed at 100,000 cycles and a further 7 have failed at 1 million cycles, then the sample would have an endurance of 100,000 cycles.

Microchip uses a more stringent criteria for endurance: no more than 2.5% of devices can have failed for the given endurance goal to have been met.

A "write cycle" is also a somewhat flexible definition since almost every customer will write the device in a different way. For example, if the customer application uses only the first three bytes of the array to store variable data, and the remainder of the array is used as a lookup table, then a write cycle will be complete when the three data bytes have been re-written to their new data state.

© 1997 Microchip Technology Inc.

A write cycle is often described as an erase/write cycle, since almost all technologies employ an "auto-erase" before the data is actually written to the array. This is also used by Microchip, but we will use the term "write cycle" since the auto-erase is invisible to, and cannot be suppressed by, the customer.

The term "data changes" is occasionally used in place of "write cycle" or "erase/write cycle." A data change will occur when an auto-erase cycle is initiated, and a second data change will occur upon the write cycle, therefore, one "erase/write cycle" is equivalent to two "data changes." The term "data change" also implies that a different type of cycling is being used than "erase/write cycle." This will be described later.

The term "write cycle" does not define under what conditions the cycling was done (unless explicitly stated) nor does it define the type of cycling that was done. The endurance cycling can be done at any number of conditions of voltage and temperature (e.g., 85°C and 5.5V, or 25°C and 5.0V) that may or may not meet with a customer's application. The cycling mode used in endurance cycling can affect the endurance of the product. All these effects will be described later.

Microchip uses the most stringent conditions that are reasonable for endurance cycling. We use byte or page mode cycling at a temperature of 85°C at 5.5V. All data not explicitly defined at other conditions is taken at these conditions.

A "read cycle" is completed when any number of bytes of data have been read from the device. For the FLOTOX-design EEPROM-based devices made by Microchip a read cycle does not affect endurance, since the data in the EEPROM is not changed. Other technologies, such as Ferroelectric technology, may have a limited number of read cycles since data is corrupted during a read.

System Design Considerations

There are a number of design considerations that the system designer can use to maximize the endurance of an EEPROM-based device, if endurance is the application's limiting factor.

As will be described in more detail later, if the designer has any control over certain environmental or operation conditions he should observe the following basic guidelines:

- Keep the application temperature as low as possible
- Keep the application voltage (or the Vcc voltage on the EEPROM-based device) as low as possible
- Write as few bytes as possible
- Use page write features wherever possible
- Write data as infrequently as possible

With these basic guidelines applied to the fullest extent, the endurance of EEPROM-based devices can be extended well beyond the guaranteed minimum endurance. Under certain very specific conditions, Microchip Serial EEPROMs have been shown to last for well over 100 million cycles.

WRITE MODES IN EEPROMS

There are three ways that EEPROM-based devices can have the entire array data contents changed. These are: byte mode, page mode, and block (or bulk) mode. Some types of devices support all three modes, others may only support one or two modes. The mode that you use to write an EEPROM-based device will affect the long term endurance of the product.

Byte mode writing is used when the contents of the array are changed one byte at a time. For many devices this is the only user-accessible write mode available. To change the entire contents of a Serial EEPROM in this way would take up to 10 seconds (using 10 ms per page on a 64K Serial EEPROM). Parallel EEPROMs such as a 28C64, which have a faster byte write time (1 ms rather than 10 ms), but no page mode would also take up to 10 seconds.

Page mode writing is a popular feature on many new designs of EEPROM memory products. This feature allows up to 8 bytes of data to be written to the memory in the same time that one byte would normally take. In this mode, the write time for a 64K Serial EEPROM can be cut from eight seconds to one second.

Block cycle is generally a test mode used by EEPROM manufacturers to make it easier to test the products. Some types of EEPROM-based products have these modes as user options (such as the ERAL and WRAL mode in 93CXX products, or the Chip Clear mode in 28CXXX products) but generally this mode is not user accessible. A block write can be done in as little as 1 ms, allowing millions of write cycles to be completed in a few hours.

A general rule to follow in choosing write modes is that the larger the number of bytes being written in a single instruction, the longer the device will last. For example, in byte mode a device might start to fail after 300,000 cycles under a particular set of conditions, but the device may last 600,000 cycles in page mode under the same conditions. In block mode the device might last 1 million cycles, under the same conditions.

The reason for this is related to the internal design of any FLOTOX EEPROM-based product. In these devices, an internal "charge pump" takes the applied Vcc voltage (typically 2.5 to 5.5V) and increases it to 15 to 25V. This voltage is required to induce the "Fowler-Nordhiem Tunneling" or "Enhanced Emission" effects that are used to program and erase EEPROM-based devices. The specific effect that is used is manufacturer-dependent. Microchip EEPROMs program by Fouler-Nordheim Tunneling.

The charge pump voltage is used to program however many EEPROM-cells are being programmed. For example, in byte mode, all the cells in a byte (8 to 16)

© 1997 Microchip Technology Inc.

are biased with the charge pump voltage. In block mode, all the cells in the array (up to 100,000, depending on the device) are biased with the charge pump voltage. The charge pump is like a current source during conditions of high load, so the voltage put out by the charge pump will be reduced slightly if more bytes are being written. If the whole array is being programmed then the charge pump voltage will be significantly reduced, but the programming current I_{PP} will be very high.

Generally, the lower the charge pump voltage the better the endurance (there is a limit since the charge pump voltage needs to be high enough to program the cell) and so the best endurance is generally achieved by using block mode cycling. Page mode is worse than block mode, but better than byte mode. Block mode is generally not a very useful cycling mode to the end user, since the data contents in the whole array will be changed to the same value (generally 00 or FF).

When Microchip tests EEPROM-based products we use byte mode cycling on devices which do not have a page mode, and page mode cycling for those that do. We encourage our customers to use page mode writing on all products which have page mode, to ensure high endurance.

ENDURANCE TESTING METHODOLOGIES

Different manufacturers use different ways to both endurance cycle and test EEPROM-based products. There is no standard for endurance cycling, or testing devices after cycling.

There are two groups of testing that Microchip performs on all products: qualification and production. Qualification testing is done for all new products, and major changes to a product or manufacturing process. Production testing is done on all devices shipped to customers.

Qualification testing at Microchip is used to test the reliability of new products, and to guarantee that the device is reliable. A great deal of testing is done, including endurance testing on all EEPROM-based products. Endurance cycling is done at the maximum rated data book value, generally 85°C or 125°C. After the rated number of cycles (10,000, 100,000, or 1,000,000) the sample (around 300 from multiple wafer lots) is tested to a full production test program. After endurance, the units are subject to "data retention" to guarantee that the required 40 years of data retention will be achieved, after the maximum number of cycles has been completed.

Endurance cycling is done under the conditions previously described, and the data retention test is performed after this. After the data retention stress is completed (which takes six weeks) the devices are tested again, to confirm the functionality of the device to all data sheet parameters.

No more than 2.5% failures are allowed after endurance, and a failure rate higher than 100 FITs after 1000 hours of data retention stress (equivalent to more than 10 years at 55°C) is unacceptable.

Production testing is done by Microchip on all devices shipped to a customer. Production testing begins immediately after a wafer lot is finished being processed, continuing in various stages until the devices are shipped to a customer.

The first tests that are done on EEPROM-based products at Microchip are called wafersort. They are done before the wafer is cut up into dice for assembly. There is a series of tests which include large numbers of write cycles (up to 5000) to ensure reliability by weeding out weak devices so they never get shipped.

After assembly full testing is done which includes further write cycles across the guaranteed temperature range, to ensure device functionality at the temperature extremes. After the normal production testing a sample of 128 units is taken from every wafer lot, and cycled to 10,000 cycles (Parallel EEPROMs) or 1,000,000 cycles (Serial EEPROMs and EEPROM microcontrollers) at 85°C using the conditions already described. After endurance testing the devices are tested for functionality at 85°C.

Any sample of Parallel EEPROMs which shows significant failures at the end of cycling causes the entire wafer lot to be pre-cycled prior to production testing. Serial EEPROMs and EEPROM-based microcontrollers do not receive the same disposition. Lots with significant failures come under scrutiny and full failure analysis with corrective actions is done. This is, however, a rare occurrence.

The testing that Microchip does is unique. Manufacturers will generally do different testing from each other. Microchip firmly believes that our testing ensures excellent quality and reliability.

THE EFFECT OF TEMPERATURE ON EEPROM ENDURANCE

The temperature at which cycling is done will affect the number of write cycles that can be executed before the device fails. The higher the temperature, the worse the endurance will be. Generally, and approximately, a device which fails at 10 million cycles at 25°C will fail at 2 million cycles at 85°C and 1 million cycles at 125°C. The reasons for this are not conclusive (although there is much technical literature supporting one theory or another) but it is apparent that the failure mode of EEPROM cells (generally considered to be electron trapping in the tunnel dielectric causing shielding and dielectric breakdown) is strongly dependent on temperature.

Data taken by Microchip suggests that if the typical failure of an EEPROM-based device is 10 million cycles at 25°C, the mean failure will occur at other temperatures according to the following table:

© 1997 Microchip Technology Inc.

TABLE 1: TEMPERATURE MEAN FAILURE

Write Cycle Temperature	Mean Failure (Cycles)
-40°C	37.1 million
0°C	16.7 million
25°C	10.0 million
40°C	7.4 million
55°C	5.4 million
70°C	4.0 million
85°C	2.9 million
100°C	2.2 million
125°C	1.3 million

This data was taken on Microchip FLOTOX Fowler-Nordhiem Tunneling EEPROMs and formed a part of the data set used to create the Total Endurance™ disk. Other technologies (such as FLOTOX Enhanced Emission or Ferroelectric technologies) may have different characteristics.

As is clearly seen, any cycling done at 25°C can be misleading in the extreme if the application requires a device that can be cycled 10 million times at, say, 55°C.

THE EFFECT OF VOLTAGE ON EEPROM ENDURANCE

The voltage at which a device is written can also affect the endurance. This is simply because the charge pump (used to program and erase EEPROM cells) is more powerful at higher voltages. As has already been described, a higher programming voltage will reduce the endurance of an EEPROM cell, and a stronger charge pump will produce a higher voltage.

Data taken by Microchip suggests that if typical failure of an EEPROM-based device is 1 million cycles when endurance cycling is done at 5.5V, mean failure occurs at other temperatures according to the following table:

TABLE 2: VOLTAGE MEAN FAILURE

Endurance Cycling Voltage	Mean Failure (Cycles)
5.5V	1.0 Million
5.0V	1.2 Million
4.5V	1.4 Million
4.0V	1.7 Million
3.5V	2.0 Million
3.0V	2.4 Million
2.5V	2.8 Million
2.0V	3.3 Million

This data was taken on Microchip FLOTOX Fowler-Nordhiem Tunneling EEPROMs and formed a part of the data set used to create the Total Endurance disk. Other technologies (such as FLOTOX Enhanced Emission or Ferroelectric technologies) may have different characteristics.

THE EFFECT OF WRITE MODE ON EEPROM ENDURANCE

As has been discussed there are three basic ways of writing data to an EEPROM-based device:

- Byte mode
- Page mode
- Block mode

This is related to the strength of the charge pump in applying the required programming voltages to the EEPROM cells.

Data taken by Microchip suggests that if the typical failure of an EEPROM-based device is 1 million cycles when the endurance cycling is done in byte mode, the mean failure will occur in other modes according to the following table:

TABLE 3: ENDURANCE MEAN FAILURE

Endurance Cycling Mode	Mean Failure (Cycles)
Byte	1.0 Million
Page	4.6 Million
Block	13.2 Million

This data was taken on Microchip FLOTOX Fowler-Nordhiem Tunneling EEPROMs and formed a part of the data set used to create the Total Endurance disk. Other technologies (such as FLOTOX Enhanced Emission or Ferroelectric technologies) may have different characteristics. This data was taken from a Microchip 24LC16.

As you can see, the use of the block cycle data to guarantee endurance can be misleading.

THE TOTAL ENDURANCE PREDICTIVE SOFTWARE

Microchip has a Windows®-based software model called Total Endurance. This program, based on all of the customers endurance parameters, will predict the failure level at the expected end of application life. This tool is invaluable for system designers who would like to fine-tune their application in favor of endurance. It is available now from your local Microchip distributor.

© 1997 Microchip Technology Inc.

MICROCHIP

AN602

How to get 10 Million Cycles Out of Your Microchip Serial EEPROM

Author: *David Wilkie*
Microchip Technology Inc.

INTRODUCTION

Microchip Technology Incorporated recently became the first manufacturer of Serial EEPROMs to rate the endurance of 1K to 16K 2-Wire protocols and 1K to 4K 3-Wire protocols at 10 million cycles. This gives the highest endurance available from any manufacturer on such a wide range of Serial EEPROMs. Microchip is also the world leader in endurance application support, including Total Endurance™ software, which allows customers to determine exactly what their endurance will be in their application. This application note will explain our 10 million cycle guarantee in the context of Total Endurance and the customer application.

ENDURANCE

Endurance is the term used to define the number of times an EEPROM-based device can be erased and re-written before an unacceptable failure rate has occurred. Microchip defines an unacceptable failure rate as 2.5%, which is half the "industry standard" (JEDEC) failure rate of 5%.

Microchip uses the industry standard cycling conditions to guarantee 10 million cycles. These are the same conditions used by the major EEPROM manufacturers, unless specifically stated otherwise. The conditions are 25°C block cycle at 5.0V. The data pattern that is written to the device is supplier-dependent. Microchip uses a checkerboard pattern to better simulate actual use.

10 MILLION CYCLE GUARANTEED

We take a great deal of data on the endurance of our devices. We run experiments over temperature, voltage, array size, data pattern, cycling type and other erase/write cycling conditions, to continually learn about our products. The data we take is continually fed back into our manufacturing process tracking, and is also used to periodically update our application predictive software, Total Endurance. The lessons that we learn are passed on to our customers.

Microchip has shown many times before that temperature has the strongest effect on endurance. The higher the temperature of your application is, the lower your endurance will be. For example, no EEPROM manufacturer can deliver a product that will achieve 1 million cycles at 125°C, but many can deliver 1 million cycles at room temperature. Microchip can provide 10 million cycles on all our Serial EEPROMs at room temperature, using the standard cycling conditions.

You can ensure the lowest possible failure rate at 10 million cycles by using the EEPROM in ways that are easiest on endurance. Use the lowest temperature possible. Use the lowest voltage possible. Write as few bytes as possible (the number of reads is unlimited), and use page modes if the device you are using has one (Typically all 2-wire protocols have a page mode, but 3-wire protocols do not).

The best way to determine the endurance of your application is to use the Total Endurance software. This will give accurate results over a very wide range of application-specific conditions, including array size, temperature, and voltage (the main effects on cycling).

The most important thing to remember is that endurance is application-specific. Each application will result in a different endurance level, based on the temperature, voltage, and other conditions under which the application operates. Currently Microchip is the only manufacturer to offer such a wide range of tools to help guide the user in his or her application of Serial EEPROMs.

© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN608

Converting to 24LCXXB and 93LCXX Serial EEPROMs

Author: *Nathan John*
Microchip Technology Inc.

Microchip Technology Inc. is constantly striving to give you, our customer, greater value. In the Serial EEPROM market this means creating devices that exhibit greater reliability, that consume less power, have more features and are priced aggressively with respect to our competitors. We have created the 24LCXXB and 93LCXX families of Serial EEPROMs to meet all of these goals and more. We are proud to announce that these families of EEPROMs are the world's first to be guaranteed to 10 Million Erase/Write cycles. Please see Table 1 below for a complete list of the benefits for converting from "C" designated parts from Microchip and other vendors.

For most applications, there are no hardware or software compatibility issues in converting to the "LC" parts, you can drop them into the socket and you will be off and running. A small percentage of designs could require some modification to your hardware or software. Please review Table 2 and Table 3 to ensure that the "LC" parts will work correctly in your systems.

TABLE 1: FEATURES AND BENEFITS OF CONVERSION

Feature	Benefit
Higher Endurance	Opens new applications for Serial EEPROMs Lower FIT rate at given Write Cycle point
Smaller Die	Lower Cost
Lower Voltage / Lower Power	Enables new battery powered uses Allows for lower overall system Vcc
Smaller Process Geometry	Avoids obsolescence of older technology
Schmitt Trigger Inputs (24LCXXB)	Filtering for better noise immunity
Rotated Pinout Versions (93LCXX/SN)	Conforms to alternate industry pinout
Hardware Write Protect (24LCXXB)	Enhanced data protection

TABLE 2: CONVERSION ISSUES FOR 24LCXXB

Conversion Issue	Detail
Addressing	The "LC" versions do not have active address pins and, therefore, you can only have one of these devices on the I²C™ bus.
Write Speed	The "LC" parts have a maximum write time of 10ms, while the "C" parts have a write time of 1ms.
Write Disable Voltage	The "LC" versions will write down to 1.5V, while the "C" versions will disable writing below 4.0V.
Write Protection	The "C" parts have either no write protection or protection on one half of the array, the "LC" parts all have full array protection, implemented by connecting pin 7 to Vcc.

7

© 1997 Microchip Technology Inc.

TABLE 3: CONVERSION ISSUES FOR 93LCXX

Conversion Issue	Detail
Write Speed	The "LC" versions have a maximum write time of 10ms, compared to 1ms for "C."
Write Disable Voltage	The "LC" parts will write down to 1.5V while the "C" parts will only write down to 4.0V.
Start of Write Cycle	The "LC" versions start the write cycle on the low going edge of chip select, the "C" versions start on the last data bit clock.

© 1997 Microchip Technology Inc.

AN609

Interfacing Microchip Serial EEPROMs to Motorola® 68HC11 Microcontroller

Author: *Keith Pazul*
Microchip Technology Inc.

INTRODUCTION

There are many different microcontrollers on the market today that are being used in embedded control applications. Many of these embedded control systems need non-volatile memory. Because of their small footprint, byte level flexibility, low I/O pin requirement, low power consumption and low cost, Serial EEPROMs are a popular choice for non-volatile storage.

Microchip Technology Incorporated addresses this need by offering a full line of Serial EEPROMs covering industry standard serial communication protocols for 2-wire and 3-wire communication. The theory, operation and differences of these two protocols are discussed in detail in Microchip's application note AN536. The reader should refer to AN536 if unfamiliar with 2-wire and/or 3-wire communication protocols. Serial EEPROM devices are available in a variety of densities, operational voltage ranges and packaging options.

Microchip realizes that its customer base is very broad, and because of this, different microcontrollers are used to interface to Serial EEPROMs. One of the microcontrollers used in these applications is the Motorola® 68HC11. In order to simplify the design process, Microchip has written a 68HC11 assembly code to communicate with our 2-wire and 3-wire parts that is verified and tested to function properly.

There are 13 programs written for inclusion in this application note. A listing of one of the programs is included with this application note as an example. The source code for all of the programs is available for downloading via Microchip's Bulletin Board (BBS). Users may consult the index of the *Microchip Embedded Control Handbook* for log-on instructions for the BBS. Once logged on to the Microchip BBS, select the FILE LIBRARY (download files) option from the main menu. Next, select a library. This application note is located in the MEM_APPS file library. Once in the MEM_APPS file library, the proper application note can be selected and downloaded by following the directions supplied by the BSS system. Only one file needs to be downloaded from the Microchip BBS to get all of the source code. All source code files are combined and compressed into one downloadible file called AN609.zip.

For Microchip 2-wire devices (24CXXA, 24LCXX, 24LCXXB, 24AAXX devices, excluding 24XX32 and 24XX65 devices), there are three programs to perform some of the basic communication functions.

- SRDMT2W.ASM - 2-Wire Sequential Read
- BW4MT2W.ASM - 2-Wire Sequential Write
- BW4PMT2W.ASM - 2-Wire Byte Write with Data Polling

Refer to the data sheets in the Microchip *Data Book* for the 2-wire devices listed above for explanations of sequential read, sequential write and data polling.

Microchip also offers a powerful and flexible family of products referred to as Smart Serial™. These devices use the same 2-wire interface as described above, but have added intelligence not found on Microchip's other 2-wire devices. These devices include the 24C32, 24LC32, 24AA32, 24C65, 24LC65, and 24AA65. Among Smart Serial features are: split erase/write cycle endurance (user selectable regions of the device with two different endurance ratings), a 64-byte write cache and the ability to permanently write protect part or all of the array.

- RRDMT65.ASM - Random Read from 64K Smart Serial
- SRDMT65.ASM - Sequential Read from 64K Smart Serial
- CACWMT65.ASM - Full Cache Write to 64K Smart Serial
- WR8MMT65.ASM - Sequential Byte Write to 64K Smart Serial
- WR8PMT65.ASM - Sequential Byte Write with Data Polling to 64K Smart Serial
- HEMT65.ASM - Setting of High Endurance block for 64K Smart Serial
- SECMT65.ASM - Setting Security Features for 64K Smart Serial

Refer to individual data sheets of the parts listed for the definition and explanation of the functions.

For Microchip 3-wire devices (93CXX, 93LCXX 93LCXXB, and 93AAXX devices), there are three programs included to perform some of the basic communication functions.

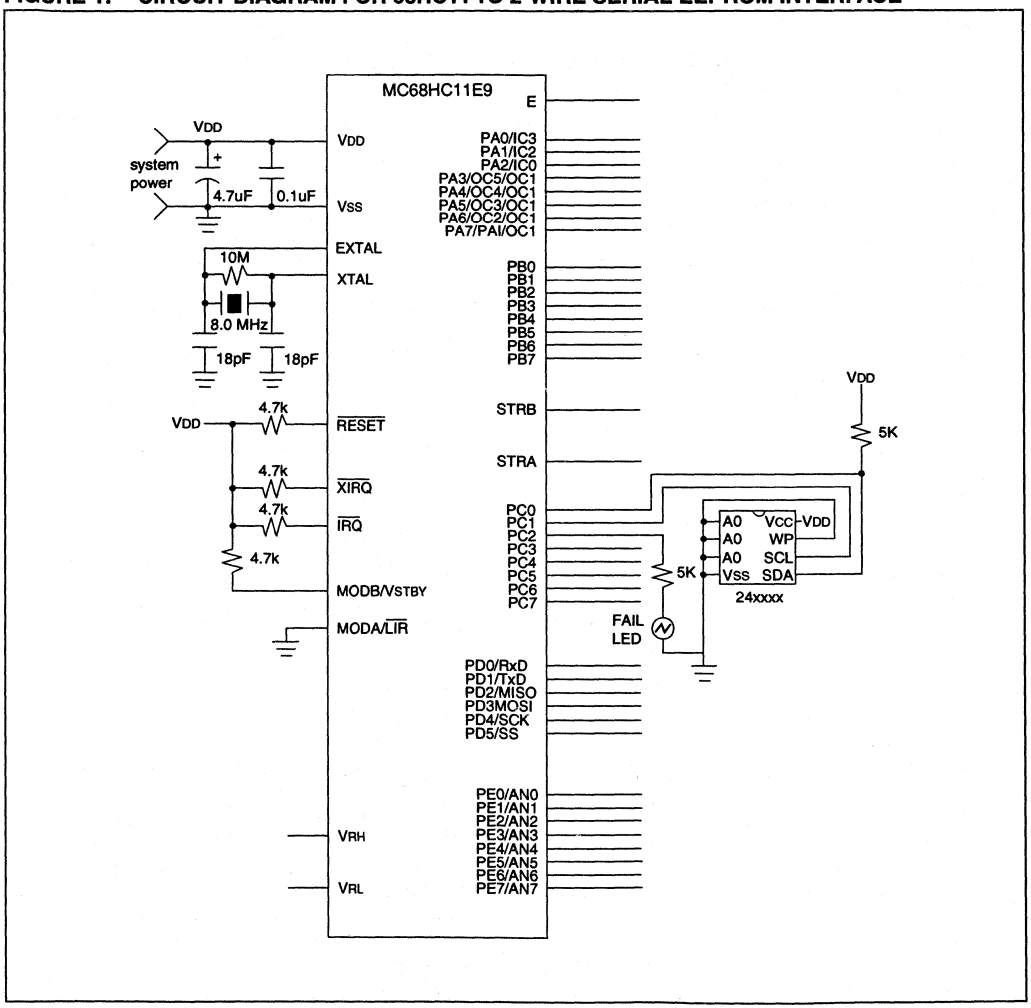
- MT4BR3W.ASM - 3-Wire Multiple Word Read
- MT4BW3W.ASM - 3-Wire Multiple Word Write
- MT4BW3PW.ASM - 3-Wire Multiple Word Write with Data Polling

© 1997 Microchip Technology Inc.

Refer to the individual data sheets of the 3-wire devices listed for explanations of multiple word read, multiple word write and data polling.

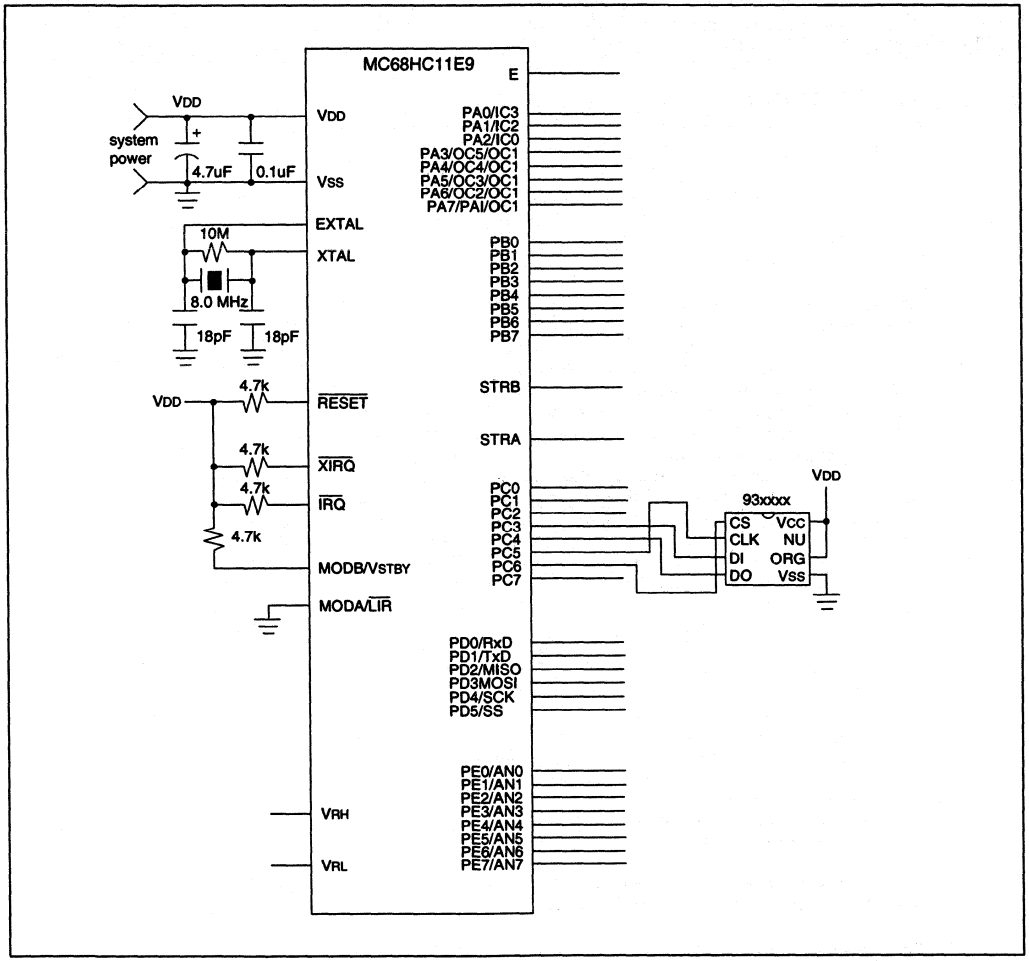
Figure 1 describes the hardware schematic for the interface between Microchip's 2-wire devices and the Motorola 68HC11E9. This schematic applies to the connection of the Smart Serial devices as well. Figure 2 describes the hardware schematic used to connect Microchip's 3-wire devices to the Motorola microcontroller. The schematics show the connections necessary between the microcontroller and the serial EEPROM, and the software was written assuming these connections.

FIGURE 1: CIRCUIT DIAGRAM FOR 68HC11 TO 2-WIRE SERIAL EEPROM INTERFACE



© 1997 Microchip Technology Inc.

FIGURE 2: CIRCUIT DIAGRAM FOR 68HC11 TO 3-WIRE SERIAL EEPROM INTERFACE



7

Please check the Microchip BBS for the latest version of the source code.
Microchip's Worldwide Web Address: www.microchip.com;
Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: SOURCE CODE

```
        ; ***************************************************************************
        ; 2-Wire Sequential Write Program (225 bytes)
        ;
        ; This program (bwr4mt2w.*) writes 4 bytes of $A5 to a Microchip 24LCxx
        ; beginning at location $10 using a Motorola 68HC11 microcontroller.
        ; This code has been verified that it writes properly to a Microchip
        ; 24LCxx serial EEPROM.
        ;
        ; This program was written using a 68CH11EVBU evaluation board.
        ; This board has a monitor program in firmware of the 68HC11 which
        ; allows single stepping, register viewing, modifying, etc. Since this
        ; is the case, the program code will be loaded into the on-chip EEPROM.
        ; This EEPROM begins at $B600. The control registers are left to their
        ; default location of $1000 and the RAM is left to its default location.
        ; RAM locations of $48-$ff are used by the monitor program and are not
        ; available for program use. Therefore, the stack pointer is set a $47
        ; and will be able to use all of the RAM to $00, and the RAM variables
        ; begin at location $100 and go up from there.  I cannot program the
        ; reset vector since it is ROM space (at $FFFE), so the way I run this
        ; program is to use the monitor program that comes with the evaluation
        ; board to set the program counter to the starting address of my user
        ; program ($B600) and begin from there.  For users who do have access
        ; to the reset vector, the label of the beginning program (in my case, it
        ; is called START) should be loaded at location $FFFE.  This program was
        ; not assembled using a Motorola assembler, but was assembled using
        ; Universal Cross-Assemblers Cross-32 Meta-Assembler.  It has the
        ; ability to assemble just about any microcontroller code.  There are
        ; certain commands that are unique to the cross-assembler.  These
        ; commands will be commented differently than other comments to
        ; be recognizable.  They will look like this:

        ;+++++++++++++++++++++++++++++++++
        ; Special cross-assembler command(s)
        ;+++++++++++++++++++++++++++++++++

        ; I do not know the exact assembler commands required to accomplish
        ; assembly using a Motorola assembler.
        ;
        ; The crystal that comes with the evaluation board is 8 Mhz.
        ; With this used as the clock input, this code will output a serial
        ; data stream at approximately 32khz.  Although this does not
        ; meet I2C spec of 100 khz, it communicate with the Microchip
        ; 24LCxx parts properly.  If a different frequency crystal is used,
        ; the code may need to be modified to meet timing specifications.


        ;*****************************************************


        ;+++++++++++++++++++++++++++++++++++++++++++++++++
0000        CPU "C:\WINC32\68HC11.TBL"  ; LOAD TABLE
0000        HOF     "MOT8"      ; Hex output is Motorola S-records
0000        PAGE     60         ; Sets # of lines in list file
                               ;  to 60 before pagebreak
        ;+++++++++++++++++++++++++++++++++++++++++++++++++
        ;
        ;*****************************************************************
        ; 68HC11 control register locations
        ;*****************************************************************
1000 =      REGBS       EQU 1000H   ; BEGINNING OF REGISTERS
```


© 1997 Microchip Technology Inc.

```
0100 =      RAMBS       EQU 100H    ; BEGINNING OF RAM VARIABLES
1007 =      DDRC        EQU REGBS+07H; DATA DIRECTION REG FOR PORT C
1003 =      PORTC       EQU REGBS+03H; PORT C DATA REGISTER
0003 =      PCOFF       EQU 03H     ; OFFSET FROM CONTROL REG BEG.
        ;*****************************************************************
        ;*****************************************************************
        ; User defined constants
        ;*****************************************************************
0003 =      CHIDHI  EQU 00000011B; SET BOTH CLK AND DATA HI
0002 =      CHIDLO  EQU 00000010B; SET CLK HI AND DATA LO
0001 =      CLODHI  EQU 00000001B; SET CLK LO AND DATA HI
0000 =      CLODLO  EQU 00000000B; SET CLK AND DATA BOTH LO
0001 =      DIMASK  EQU 00000001B; BIT MASK FOR DATA IN BIT
0080 =      DOMASK  EQU 10000000B; BIT MASK FOR DATA OUT BIT
0001 =      SDAMASK EQU 00000001B; BIT MASK FOR SERIAL DATA
0002 =      SCKMASK EQU 00000010B; BIT MASK FOR SERIAL CLOCK
0004 =      LEDMASK EQU 00000100B; BIT MASK FOR ACK FAILED LED
        ;*****************************************************************

0000 8E0047       LDS#0047H; STACK POINTER BEGINS AT $47

0100        ORG 100H        ; RAM variables begin at 100h

        ;+++++++++++++++++++++++++++++++++++++++++++++++++
        ; DFS is a Universal Cross-Assembler directive that stands for define
        ; storage.  1 is for byte, 2 is for word, 4 is for long word.  These are
        ; the user defined RAM variables.

0100        TXBUFF  DFS 1   ; 100H
0101        EE_IN   DFS 1   ; 101H
0102        ADDR    DFS 1   ; 102H
0103        RXBUFF  DFS 1   ; 103H
0104        BYTECNT DFS 1   ; 104H
        ;+++++++++++++++++++++++++++++++++++++++++++++++++

        ;*****************************************************************
        ; Program code cannot be placed in ROM for eval board because
        ; eval board firmware is loaded in ROM.  Program code will be
        ; loaded in EEPROM (which is 512 bytes and begins at B600h).

        ; Serial data (SDA) is located on port c, pin 0
        ; Serial clock (SCK) is located on port c, pin 1
        ; An acknowledge fail LED is connected to port c, pin 2.  When
        ; the ack is not low like is should be, the ack failed LED is
        ; illuminated.

B600        ORG 0B600H
        ;*****************************************************************
        ;
        ; This is the main portion of the code.  It is where the reset
        ; vector should set program counter to.
        ;
        ;*****************************************************************

B600        START
B600 8610       LDAA#00010000B; ADDRESS OF MEMORY TO BEGIN
B602 B70102     STAAADDR    ;  WRITING DATA
B605 8604       LDAA#4      ; NUMBER OF BYTES TO WRITE OUT
B607 B70104     STAABYTECNT;
B60A CE1000         LDX #REGBS  ; LOAD $1000 INTO X INDEX REG


B60D BDB639     JSR STRTBIT ; GOTO STRTBIT SUBROUTINE
B610 86A0       LDAA#10100000B; LOAD CONTROL BYTE INTO
B612 B70100     STAATXBUFF  ;  TXBUFF FOR OUTPUT TO SEEPROM
```


© 1997 Microchip Technology Inc.

```
B615 BDB6A2    JSR TXBYTE  ; OUTPUT 1 BYTE TO SEEPROM

B618 B60102        LDAAADDR    ; GET ADDRESS AND LOAD IN
B61B B70100        STAATXBUFF  ;  TXBUFF FOR OUTPUT
B61E BDB6A2        JSR TXBYTE  ; OUTPUT 1 BYTE TO SEEPROM

B621 F60104LDABBYTECNT
B624 86A5 NEXTWRLDAA#10100101B; DATA BYTE TO OUTPUT IS A5H
B626 B70100STAATXBUFF      ;
B629 37     PSHB           ; PUSH DATA BYTE COUNTER TO
                           ;  STACK
B62A BDB6A2JSR TXBYTE      ; OUTPUT 1 BYTE TO SEEPROM
B62D 33     PULB           ; PULL DATA BYTE COUNTER FROM
                           ;  STACK
B62E 5A     DECB           ; HAVE WE OUTPUT CORRECT # OF
                           ;  DATA BYTES?
B62F C100   CMPB#00H       ;
B631 26F1   BNE NEXTWR     ; NO, THEN SEND NEXT BIT

B633 BDB653JSR STOPBIT     ; SEND STOP BIT TO BEGIN
                           ; INTERNAL WRITE CYCLE

B636 7EB600JMP START       ; START OVER AGAIN
       ;*********************************************************
       ;
       ;     Start bit output subroutine
       ;
       ;*********************************************************
B639       STRTBIT
B639 8607      LDAA#00000111B;
B63B B71007    STAADDRC    ; PORT C ALL INPUTS EXCEPT BITS
                           ;  0,1,2

B63E 8603      LDAA#CHIDHI; SET SCLK AND SDATA HI
B640 B71003    STAAPORTC   ;
B643 01        NOP         ; OBEY PROPER START BIT SETUP
                           ;  TIME
B644 01        NOP         ;
B645 01        NOP         ;
B646 01        NOP         ;
B647 01        NOP         ;
B648 1D0301    BCLRPCOFF,X,SDAMASK; SET DATA LOW FOR STOP BIT
B64B 01        NOP         ; OBEY PROPER START BIT HOLD
                           ;  TIME
B64C 01        NOP         ;
B64D 01        NOP         ;
B64E 01        NOP         ;
B64F 1D0302    BCLRPCOFF,X,SCKMAS; SET CLK LO
B652 39        RTS         ; END START BIT SUBROUTINE
       ;*************************************************************



       ;*************************************************************
       ;
       ;     Stop bit output subroutine
       ;
       ;*************************************************************

B653       STOPBIT
B653 8607      LDAA#00000111B;

B655 B71007    STAADDRC    ; PORT C ALL INPUTS EXCEPT FOR
                           ;  BITS 0,1
B658 1D0301    BCLRPCOFF,X,SDAMASK; MAKE SURE DATA BIT IS LOW
B65B 1C0302    BSETPCOFF,X,SCKMASK; CLK BIT HI
```


© 1997 Microchip Technology Inc.

```
B65E 01         NOP             ; OBEY PROPER STOP BIT SETUP
                                ;  TIME
B65F 01         NOP             ;
B660 01         NOP             ;
B661 01         NOP             ;
B662 1C0301     BSETPCOFF,X,SDAMASK; DATA BIT HI CAUSES STOP BIT
B665 39         RTS             ; END STOP BIT SUBROUTINE
        ;*************************************************************
        ;   Remember to wait internal write cycle time or acknowledge
        ;   pole here until writing is complete before beginning next
        ;   write to part.
        ;*************************************************************



        ;*************************************************************
        ;
        ;   This routine reads in one bit from the data line
        ;
        ;*************************************************************

B666        INBIT
B666 8606       LDAA#00000110B; SET SDATA AS INPUT AND KEEP
B668 B71007     STAADDRC    ;  SCLK AS OUTPUT

B66B 7F0101     CLR EE_IN   ;  GUESS INPUT IS A 0

B66E 1C0302     BSETPCOFF,X,SCKMASK; SET CLK BIT HI
B671 01         NOP             ; WAIT TO READ INPUT
B672 B61003     LDAAPORTC   ; GET INPUT FROM SDATA
B675 1D0302     BCLRPCOFF,X,SCKMASK; BRING CLK LO AFTER PORT READ
B678 8501       BITA#DIMASK; SEE IF INPUT IS 1 OR 0
B67A 2705       BEQ DONEIN  ; INPUT IS A ZERO

B67C 86FF       LDAA#0FFH   ; INPUT BIT IS ACTUALLY A 1
B67E B70101     STAAEE_IN   ; STORE BACK IN EE_IN

B681        DONEIN
B681 39         RTS
        ;*************************************************************



        ;*************************************************************
        ;
        ;   This routine writes out one bit to the sdata line
        ;
        ;*************************************************************

B682        OUTBIT
B682 8607       LDAA#00000111B;
B684 B71007     STAADDRC    ; BOTH CLK AND DATA ARE
                            ; OUTPUTS

B687 B60100     LDAATXBUFF  ; WHAT ARE WE TRYING TO
                            ;  OUTPUT,
B68A 8580       BITA#DOMASK;  A 1 OR 0?
B68C 2705       BEQ LOWOUT  ; EE_OUT BIT IS 0
B68E 1C0301     BSETPCOFF,X,SDAMASK; HIGH NEEDS TO BE SENT OUT

B691 2003       BRA CONTOUT
B693        LOWOUT
B693 1D030      BCLRPCOFF,X,SDAMASK; SEND OUT A LOW
B696 1C0302
            CONTOUTBSETPCOFF,X,SCKMASK; SET CLOCK BIT
B699 01         NOP             ; WAIT PROPER SCLK HI TIME
```

7

```
B69A 01         NOP         ;
B69B 1D0302     BCLRPCOFF,X,SCKMASK; CLR CLOCK BIT AND THEN
B69E 1D0301     BCLRPCOFF,X,SDAMASK; SET DATA BIT TO 0 FOR NEXT TX
B6A1 39         RTS
        ;*************************************************************

        ;*************************************************************
        ;
        ;   This routine outputs 1 byte of data out the sdata pin
        ;
        ;*************************************************************

B6A2        TXBYTE
B6A2 C608       LDAB#8      ; SET BIT COUNTER

B6A4 BDB682  TXBITJSROUTBIT; SEND 1 BIT
B6A7 790100     ROL TXBUFF  ; GET NEXT BIT READY TO XMIT
B6AA 5A         DECB
B6AB C100       CMPB#00H    ; HAVE WE OUTPUT ALL 8 BITS
B6AD 26F5       BNE TXBIT   ; NO, THEN SEND NEXT BIT

; GET ACK BIT AND TEST IF IT IS LOW.  IF NOT, TURN ON ACK FAIL LED
B6AF BDB666     JSR INBIT   ; RECEIVE ACK BIT
B6B2 B60101     LDAAEE_IN   ;
B6B5 8501       BITA#DIMASK; TEST IF INPUT IS 0 OR 1
B6B7 2703       BEQ DONETX  ; IF ACK IS LOW, GO TO END OF TXBYTE
B6B9 1C0304     BSETPCOFF,X,LEDMASK; TURN ON ACK FAILED LED
B6BC 39     DONETXRTS
        ;*************************************************************


        ;*************************************************************
        ;
        ;   This subroutine receives 1 byte from SEEPROM
        ;
        ;*************************************************************

B6BD        RXBYTE
B6BD C608       LDAB#8      ; LOAD BIT COUNTER IN ACCB

B6BF 790103RXBITROLRXBUFF ; GET READY TO RECEIVE NEXT BIT
B6C2 B60103     LDAARXBUFF  ; GUESS THAT INPUT BIT IS 0
B6C5 84FE       ANDA#11111110B;
B6C7 B70103     STAARXBUFF  ; WRITE ACCA BACK TO RXBUFF

B6CA BDB666     JSR INBIT   ; RECIEIVES 1 BIT
B6CD B60101     LDAAEE_IN   ; IS IN BIT A 1 OR A 0?
B6D0 8501       BITA#DIMASK;
B6D2 2708       BEQ CONTRX  ;π

B6D4 B60103     LDAARXBUFF  ; GET RXBUFF INPUT
B6D7 8A01       ORAA#00000001B; SET INPUT BIT TO 1

B6D9 B70103     STAARXBUFF  ;

B6DC 5A     CONTRXDECB
B6DD C100       CMPB#00H    ; HAVE WE OUTPUT ALL 8 BITS
B6DF 26DE       BNE RXBIT   ; NO, THEN SEND NEXT BIT

B6E1 39         RTS
        ;*************************************************************

        ;+++++++++++++++++++++++++++++++++++++++++++++
        ; This command tells the cross-assembler that code starts
        ; at the location of START

B600            END START
```


© 1997 Microchip Technology Inc.

Using the 24LC21 Dual Mode Serial EEPROM

Author: *Bruce Negley*
Microchip Technology Inc.

INTRODUCTION

The Microchip Technology Inc. 24LC21 is a 1K-bit (128 x 8) dual mode serial EEPROM that was developed primarily for use in computer monitors. This part was developed with inputs from several computer monitor manufacturers, in accordance with the VESA® (Video Electronics Standards Association) monitor committee. This committee has developed a serial communication protocol called Data Display Channel (DDC™) which was created to eliminate the need to change dip switches when configuring a new system or adding a new monitor or video card. The 24LC21 device is used in the monitor to store and transmit the EDID (extended display ID) table which contains all set-up parameters needed by the video card to operate with a particular monitor. With this system, the user can now plug any compatible monitor into any compatible graphics board and the graphics board will automatically know what type of monitor is being used and configure itself accordingly. This automatic configuration is the cornerstone for Microsoft®'s 'Plug and Play' capability being built into the new 'Windows 95™' release.

DEVICE OPERATION

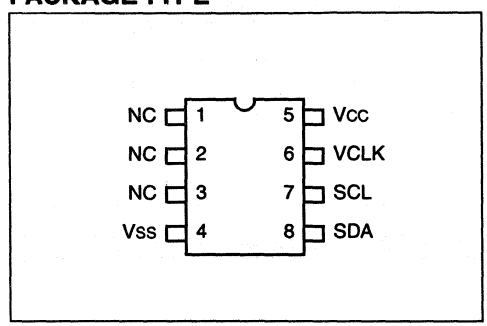
The 24LC21 can operate in two modes of operation. These two modes of operation are the transmit only mode and bi-directional mode. Upon power-up, the device will always be in the transmit-only mode. Transmit only mode is also referred to as DDC1 mode. The transmit only mode only allows the video card to read the contents of the 24LC21 in a sequential manner, one bit at a time. Writing to the device is not possible in transmit only mode.

The device will automatically transition to the bi-directional mode whenever a falling edge is seen on the SCL pin. Bi-directional mode is also referred to as DDC2 mode, and is implemented as the standard I^2C™ protocol. This allows a controller to read and write specific addresses in the device like a standard I^2C Serial EEPROM device. Once the device has transitioned to the bi-directional mode, there is no way to return to transmit only mode other than to reset (power-down) the device.

TRANSMIT ONLY MODE (DDC1)

The 24LC21 will always power-up in the transmit only mode. In this mode, the 24LC21 will output one bit of data at the SDA pin for every rising edge on the VCLK pin. The data will be transmitted in 8-bit words, with each word followed by a 9th null bit. This null bit will always be high. A timing diagram for transmit only mode is shown in Figure 1. As long as VCLK is present and no falling edges on SCL are received, the 24LC21 will repeatedly cycle through the entire memory array.

PACKAGE TYPE



Upon power-up, the device will not output valid data until it has been initialized. This initialization procedure (Figure 1) data will not be available until after the first 9 clocks are sent to the device. The exact memory location that the 24LC21 begins to transmit data is unknown at power-up, and the initialization procedure only initializes the device, not the starting address or bit location. In order to for a controller to determine what address is being read, a 'framing' or 'syncing' procedure must be executed by the video card.

© 1997 Microchip Technology Inc.

A framing procedure involves looking for the header portion of the EDID table which is a byte of 00H followed by 6 bytes of FFH and another byte of 00H. A framing routine would continue to clock data from the 24LC21 until this unique header has been found. At this point, the current location in the EDID table has been determined and the controller has now synchronized itself with the device. Care must be taken while using the device in the transmit only mode to prevent noise on the SCL pin, as a falling edge seen on this pin will immediately send the part into the bi-directional mode. In a DDC1-only monitor, SCL is not connected to the VGA connector, but must still be terminated to Vcc through a pullup resistor.

FIGURE 1: TRANSMIT ONLY MODE

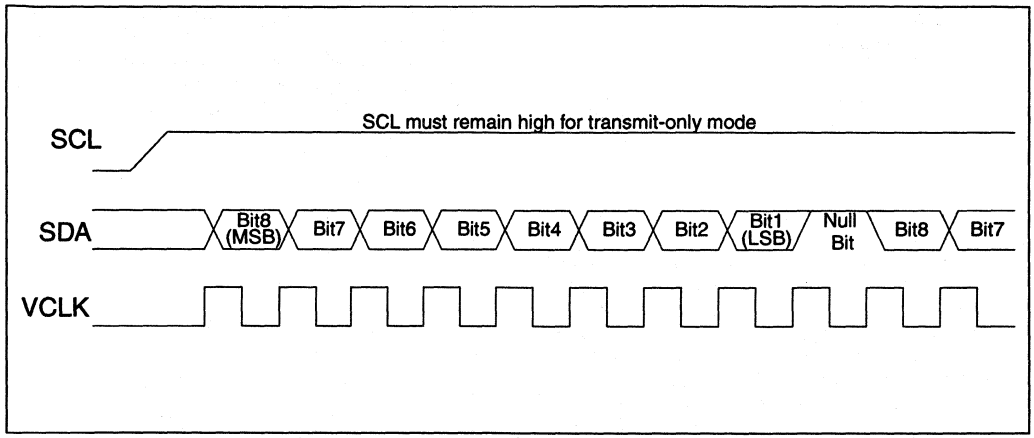
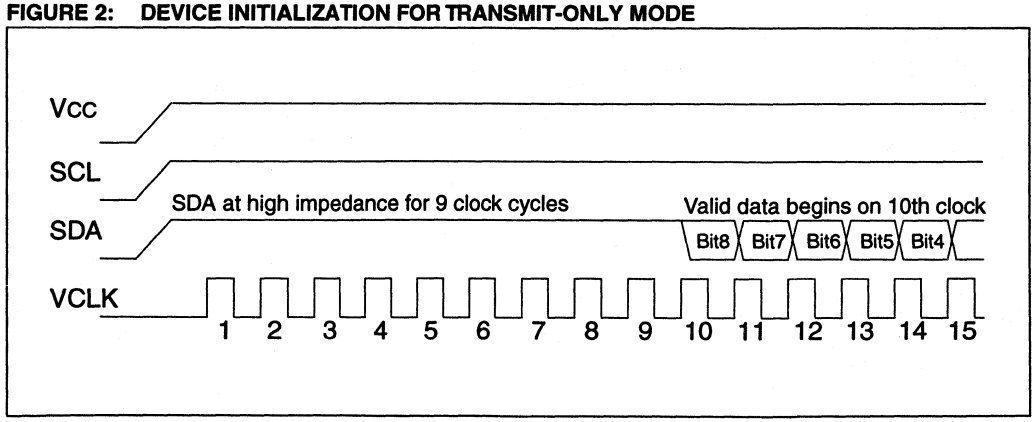


FIGURE 2: DEVICE INITIALIZATION FOR TRANSMIT-ONLY MODE



© 1997 Microchip Technology Inc.

BI-DIRECTIONAL MODE (DDC2)

Bi-directional mode is essentially the standard I^2C protocol and allows the controller to read and write to the device. The 24LC21 supports byte and page writes and byte and sequential reads in the bi-directional mode. This mode will be used primarily before the monitor leaves the factory to load the EDID table into the device, but it also provides a means of updating the table if necessary. It is also used for faster (up to 100 kHz) data transmission, or transmission of only specific requested data in a DDC2 system. (The I^2C protocol allows the host to request data from a specific portion of the EDID table rather than waiting for the entire table.) When writing to the device, the VCLK pin must be held high while the write command is being loaded or the write will be aborted and no data will be written. Note that this is the opposite of the 24LC01B, where the WP pin must be held low for the device to be written.

EDID TABLE

The EDID table is the Extended Display ID table, specified by VESA, that will be stored in the 24LC21 and contains information about what type of display it is and the capabilities of the display. The basic EDID table consists of 128 bytes of data. A breakdown of the table is shown below in Table 1. A complete description of the table can be found in the VESA DDC Specification.

TABLE 1: EDID TABLE DESCRIPTION

Bytes	Description
8	Header
10	Vendor/Product Description
2	EDID Version/Revision
15	Basic Display Parameters/Features
19	Established/Standard Timings
72	Detailed Timing Descriptions (18 bytes each)
1	Extension Flag
1	Checksum

7

© 1997 Microchip Technology Inc.

USING THE 24LC21 IN A SYSTEM

In order to use the 24LC21 in a monitor system, it must be programmed with a proper EDID table and then properly connected to the signals coming from the video controller card. The VESA committee has specified that the connections for DDC transmission can be part of the standard 15-pin VGA connector. A table of pinouts for this connector are shown in Table 2. Signals that pertain to the use of the 24LC21 are highlighted.

Programming of the 24LC21 can be accomplished via Microchip Technology's SEEVAL programming and evaluation system or by any final test system at the customer site which can communicate over the I^2C bus.

TABLE 2: VGA CONNECTOR DESCRIPTION

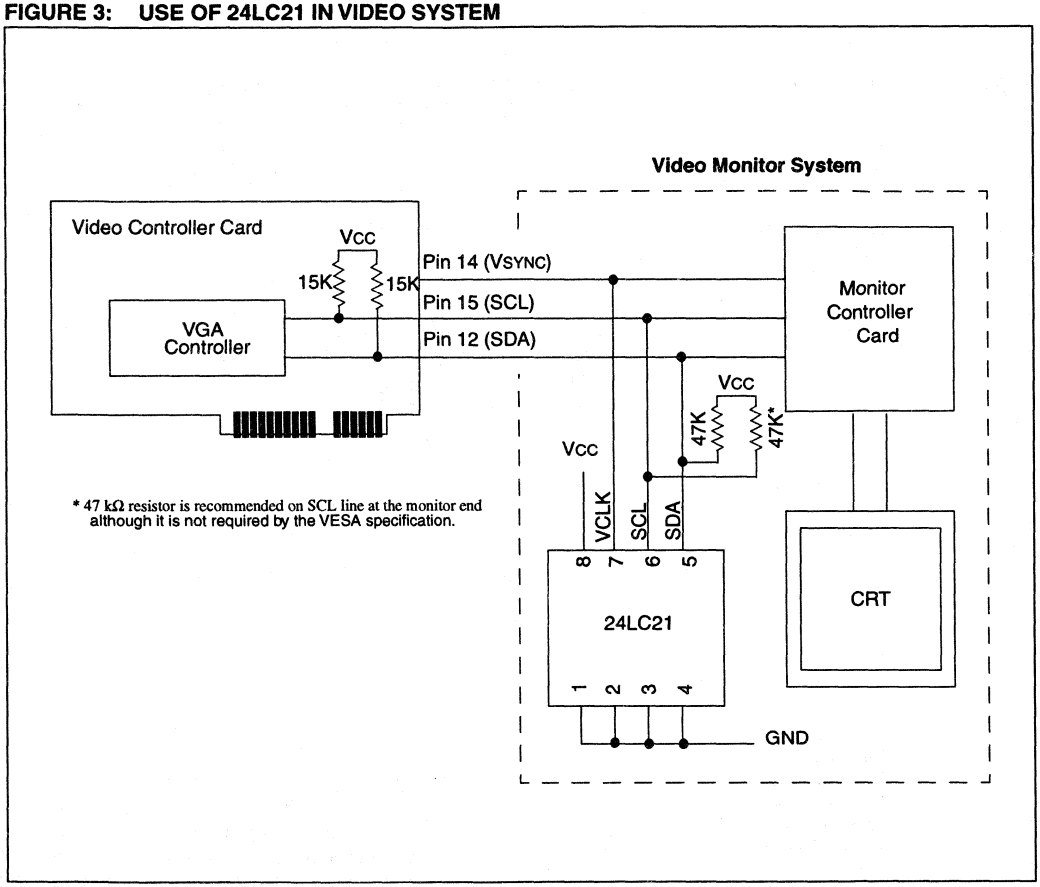
Pin	Standard VGA	DDC1 Host	DDC2 Host	DDC1.2 Display
1	Red Video	Red Video	Red Video	Red Video
2	Green Video	Green Video	Green Video	Green Video
3	Blue Video	Blue Video	Blue Video	Blue Video
4	Monitor ID Bit2	Monitor ID Bit2	Monitor ID Bit2	Return
5	Test (Ground)	Return	Return	Return
6	Red Video Return	Red Video Return	Red Video Return	Red Video Return
7	Green Video Return	Green Video Return	Green Video Return	Green Video Return
8	Blue Video Return	Blue Video Return	Blue Video Return	Blue Video Return
9	No Connection	+5V Supply (optional)	+5V Supply (optional)	+5V Supply (optional)
10	Sync Return	Sync Return	Sync Return	Sync Return
11	Monitor ID Bit0	Monitor ID Bit0	Monitor ID Bit0	Optional
12	Monitor ID Bit1	**Data from Display (SDA)**	**Bi-directional Data (SDA)**	**Bi-directional Data (SDA)**
13	Horizontal Sync	Horizontal Sync	Horizontal Sync	Horizontal Sync
14	Vertical Sync	**Vertical Sync (VCLK)**	Vertical Sync	**Vertical Sync (VCLK)**
15	Monitor ID Bit3	Monitor ID Bit3	**Data Clock (SCL)**	**Data Clock (SCL)**

© 1997 Microchip Technology Inc.

SYSTEM CONFIGURATION

A typical system configuration is shown below. The DDC specification states that a 47 kΩ pull-up resistor is required on the SDA line at the monitor end. It also states that a 15K pullup resistor is needed on both the SCL and SDA lines at the video controller end.

FIGURE 3: USE OF 24LC21 IN VIDEO SYSTEM

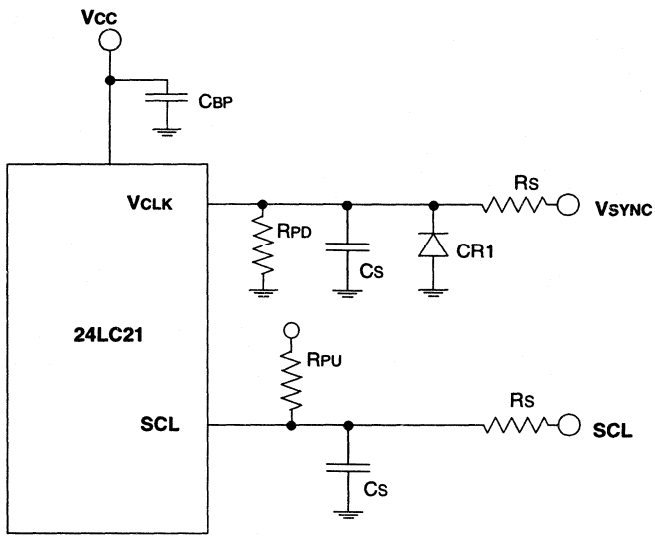


7

POTENTIAL PROBLEMS CAUSED BY NOISE IN A VIDEO SYSTEM

Because the typical application for the 24LC21 is in a computer monitor where electronic noise is prevalent, some precautions may need to be made in order for this device (or any other CMOS device) to work properly. The diagram below (Figure 4) shows a filter circuit that can be used to reduce the amount of noise seen by the device on the SCL and VCLK pins.

FIGURE 4: RECOMMENDED FILTER CIRCUIT FOR MONITOR APPLICATIONS



CBP = 1 µF
RS = 100 - 300 Ω
RPD = 4.7 kΩ
CS = 100 - 1000 pF
RPU = 47 kΩ

CBP	Bypass capacitor
RPD	Can be as a termination resistor on VGA cable. Also will discharge the series capacitor going to the MCU and horizontal/Vertical processor.
CS and RS on VCLK	Acts as low pass filter to clean-up noise on VSYNC line
CS and RS on SCL	Acts as low pass filter to clean-up noise and dampen power transient spikes that may cause accidental mode switching from DDC1 to DDC2
RPU on SCL	Keeps SCL pulled high, although a high enough value is used that the host will not power the 24LC21. Lets the 24LC21 reset when the monitor power is turned off
CR1	Eliminates undershoot on VSYNC

© 1997 Microchip Technology Inc.

MICROCHIP

AN613

Using Microchip 93 Series Serial EEPROMs with Microcontroller SPI™ Ports

Author: *Keith Pazul*
Microchip Technology Inc.

INTRODUCTION

Systems requiring embedded control are becoming more and more sophisticated, with microcontrollers required to control these systems increasing in complexity. Many microcontrollers today are being designed with built-in serial communication capability to be able to easily access other features that are not built in to the microcontroller itself. Devices like EEPROMs, A/Ds, D/As, LCDs etc. are all being built with a serial interface to reduce cost, size, pin count, and board area. There are many different serial interfaces on the market used to interface peripherals (I^2C™, Microwire®, SPI, for example). One of the serial interfaces that is gaining in popularity is SPI (Serial Peripheral Interface). It is becoming more popular because of its communication speed, simultaneous full-duplex communication, and ease of programming.

Microchip PIC16C64/74 microcontrollers have a built-in serial port that can be configured as an SPI port. Currently, the Microchip Serial EEPROM product line does not support SPI interface Serial EEPROMs, however it is possible to use the 93 series devices on the SPI port. Any version of Microchip's 93 series devices can be communicated with via the SPI port of a PIC16C64/74. The code for this application note is written for a Microchip 93LC56/66, but talking to other 93 series devices can be accommodated with minor code changes. See the *Theory of Operation* section of this application note for more details on how this is accomplished. This code was verified by downloading to Microchip's PICMASTER™ in-circuit emulator (run at full speed with a 10 MHz crystal) and tested to make sure it writes (polling ready/busy pin to verify write cycle completion) and reads properly. A schematic (Figure 1) is included in this application note to describe exactly how the 93LC56/66 was connected to the PIC16C64/74.

The SPI interface was popularized by the Motorola 68HCXX microcontrollers. Microchip receives many requests for Motorola assembly code that uses the SPI port of the 68HC11 or 68HC05 to talk to Microchip serial EEPROMs. Because of this, Microchip has written 68HC11 assembly code to communicate with its 93 series devices via its SPI port. The program was downloaded to a 68HC11 evaluation board and tested to make sure it writes (polling ready/busy pin to verify write cycle completion) and reads properly. A schematic (Figure 2) is included in this application note to describe exactly how the 93LC56/66 was connected to the microcontroller.

THEORY OF OPERATION

To use an SPI port to communicate with Microchip's 93 series Serial EEPROMs, the bytes to be output to the 93XXXX must be aligned such that the LSB of the address is the 8th bit (LSB) of a byte to be output. From there, the bits should fill the byte from right to left consecutively. If more than 8 bits are required, then two bytes will be required to be output. This same method will work for any 93 series device. A 93LC66 was chosen as the device to write this application note code for, so the following example will be for that particular device. The theory explained below will work for any 93 series device.

Since more than 8 bits are required to control a 93LC66, two consecutive bytes are required.

High Byte (where the start bit, op code bits, and address MSB reside)

| 0 | 0 | 0 | 0 | SB | OP1 | OP0 | A8 |

The High Byte is configured in the following format: SB is the start bit, OP1 is op code MSB, OP0 is op code LSB, and A8 is the 9th address bit that is required to address 512 bytes. The CS can be set before the byte is output because the leading 0's output to the 93xxxx prevent a start bit from being recognized by the 93xxxx until the first high bit is sent.

Low Byte (8 address LSBs)

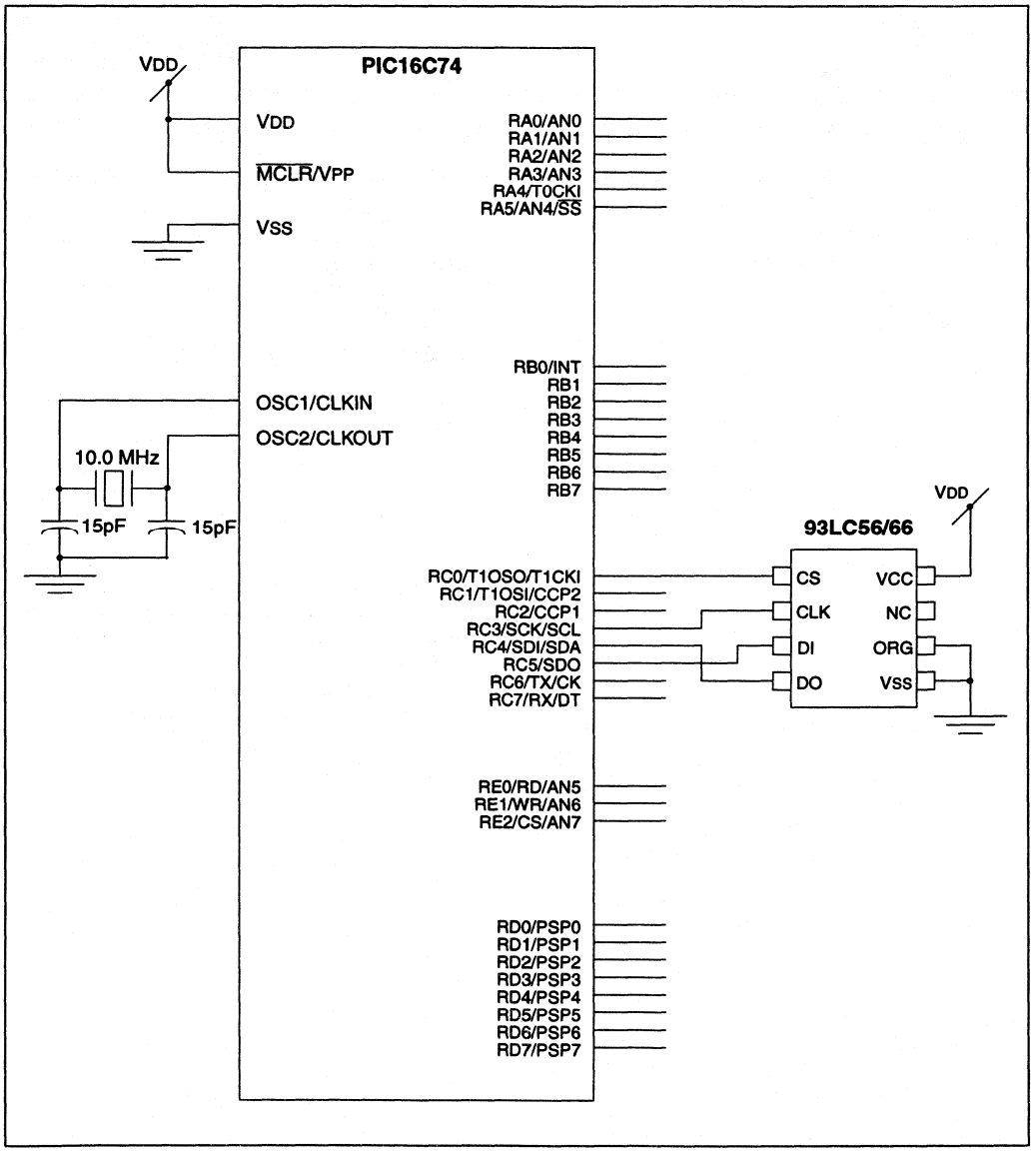
| A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 |

The Low Byte contains A7-A0, which are the rest of the address bits required to access 512 bytes.

Data output from master MUST be set up on the falling edge of the clock so that it can be read from the 93XXXX on the next rising edge. Receiving data from the 93XXXX MUST also happen on the falling edge of the clock because the data is output from the 93XXXX on the rising edge of the clock. THIS REQUIRES THE CLOCK PHASE BIT OF THE SPI PORT TO BE OPPOSITE FOR RECEIVING THAN IT IS FOR TRANSMITTING. The clock phase needs to be toggled between 0 for transmitting data and 1 for receiving data. See source code for the exact spot where the clock phase bit needs to be changed.

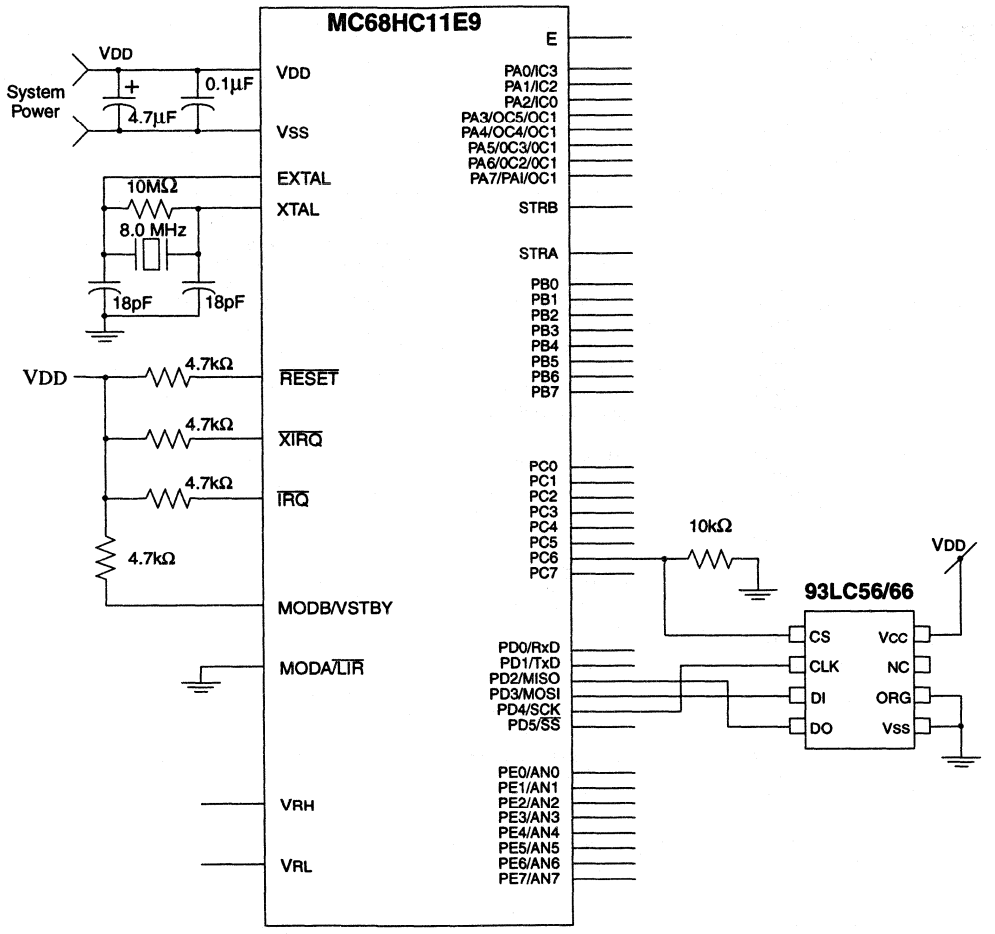
© 1997 Microchip Technology Inc.

FIGURE 1: PIC16C74 TO 93LC56/66 SCHEMATIC



© 1997 Microchip Technology Inc.

FIGURE 2: MOTOROLA 68HC11 TO MICROCHIP 93LC56/66 SCHEMATIC



Note: A pull down resistor is included on the CS pin. It deselects the 93LC56/66, always, except for when CS is driven high by the 68HC11.

7

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: PIC16C64/74 SOURCE CODE

```
MPASM 01.40.01 Intermediate  C74S66DP.ASM   4-7-1997  15:42:16         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001 ;*************************************************
                     00002 ;
                     00003 ;       To use the SPI port to communicate with 3-wire
                     00004 ;       devices, the bytes to be output must be aligned
                     00005 ;       such that the LSB of the address is the 8th bit
                     00006 ;       (LSB) of a byte to be output.  From there, the
                     00007 ;       bits should fill the bits from right to left
                     00008 ;       consecutively.  This method will work for any
                     00009 ;       93xxxx device.  This particular code shows how
                     00010 ;       to fill the bytes to be output to a 93LC66 that
                     00011 ;       is configured in x8 mode.
                     00012 ;
                     00013 ;       First byte (where start bit, op codes reside)
                     00014 ;
                     00015 ;       |  0  |  0  |  0  |  0  | SB  | OP1 | OP0 | A8  |
                     00016 ;
                     00017 ;       where SB is start bit, OP1 is op code msb and
                     00018 ;       OP0 is op code lsb, and A8 is address msb.  The
                     00019 ;       leading 0's are required, otherwise the start bit
                     00020 ;       will be send prematurely and the number of clock
                     00021 ;       cycles will not be correct.
                     00022 ;
                     00023 ;       Next byte (address)
                     00024 ;
                     00025 ;       |  A7 |  A6 |  A5 |  A4 |  A3 |  A2 |  A1 | A0  |
                     00026 ;
                     00027 ;       where A8-A0 are address bits required to access 4k
                     00028 ;       of memory.
                     00029 ;
                     00030 ;       The chip select is set high before sending the first
                     00031 ;       byte out because the 93LC66 will not recognize a
                     00032 ;       start bit until both CS and DI are high on the
                     00033 ;       rising edge of a clock.
                     00034 ;
                     00035 ;       This code is written to use the 16C64/74 SPI port
                     00036 ;       to communicate with a Microchip 93LC66.  The
                     00037 ;       only other I/O line required besides the SPI port
                     00038 ;       pins is a chip select.  The ORG pin will be grounded
                     00039 ;       to set up the part in x8 mode.
                     00040 ;
                     00041 ;       PIN DESCRIPTIONS:
                     00042 ;
                     00043 ;       SCK (serial clock)      port C, bit 3
                     00044 ;       SDO (serial data out)   port C, bit 5
                     00045 ;       SDI (serial data in)    port C, bit 4
                     00046 ;       CS (chip select)        port C, bit 0
                     00047 ;
                     00048 ;       Transmits from the master MUST happen on the falling
                     00049 ;       edge of the clock so they can be read by the 93xxxx
                     00050 ;       on the next rising edge of the clock.  Receiving from
                     00051 ;       the 93xxx MUST also happen on the falling edge of the
                     00052 ;       clock because on the rising edge of the clock, the
                     00053 ;       93xxxx outputs its bit on its DO pin.  This requires
                     00054 ;       the CKP bit in the SSPCON register to be set to 1 for
```


© 1997 Microchip Technology Inc.

```
                  00055 ;       transmitting data and CKP=0 for receiving data.
                  00056 ;
                  00057 ;       This code was written by Keith Pazul on 10/5/94
                  00058 ;
                  00059 ;*****************************************************
                  00060         LIST P=16C74
                  00061 ;*****************************************************
                  00062 ;               Ram Register Definitions
                  00063 ;*****************************************************
                  00064 ;
                  00065 ; received bytes from EEPROM memory locations 10h to 13h
                  00066 ; will be stored in RAM registers 20h to 23h.
  00000024        00067 rxdata  equ     24h
  00000025        00068 txdata  equ     25h
  00000026        00069 addr    equ     26h
  00000027        00070 loops   equ     27h
  00000028        00071 loops2  equ     28h
  00000029        00072 hibyte  equ     29h
  0000002A        00073 lobyte  equ     2Ah
  0000002B        00074 datbyt  equ     2Bh
                  00075 ;*****************************************************
                  00076 ;
                  00077 ;               Other Definitions
                  00078 ;
                  00079 ;*****************************************************
                  00080 ;
                  00081 ;
                  00082 ;*****************************************************
                  00083 ;               Bit Definitions
                  00084 ;*****************************************************
                  00085 ;
  00000000        00086 cs      equ     0
  00000004        00087 sdi     equ     4
                  00088
                  00089 ;*****************************************************
                  00090
                  00091         include "p16c74.inc"            ; This is a standard include
                  00001         LIST
                  00002 ; P16C74.INC Standard Header File, Ver. 1.00 Microchip Technology, Inc.
                  00318         LIST
                  00092                                 ;  file that comes with MPASM
                  00093
0000              00094         org     0x000           ; Reset Vector
0000 282D         00095         goto    Start
                  00096 ;
                  00097 ;*************************************************************
                  00098 ;
                  00099 ;       This is the transmit/receive routine.  The received bytes
                  00100 ;       are don't cares until reading back from the array.
                  00101 ;
                  00102 ;*************************************************************
0001 0093         00103 output  movwf   SSPBUF          ; place data in buffer so it
                  00104                                 ;  can be output
0002 1683         00105 loop1   bsf     STATUS, RP0     ; specify bank 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0003 1C14         00106         btfss   SSPSTAT, BF     ; has data been received (xmit done)?
0004 2802         00107         goto    loop1           ; not done yet, keep trying
0005 1283         00108         bcf     STATUS, RP0     ; specify bank 0
0006 0813         00109         movf    SSPBUF, W       ; empty receive buffer, even if we
                  00110                                 ;  don't need received data
0007 00A4         00111         movwf   rxdata          ; put received byte into location
0008 3400         00112         retlw   0               ; return from subroutine
                  00113 ;*************************************************************
                  00114
                  00115 ;*************************************************************
                  00116 ;
```


© 1997 Microchip Technology Inc.

```
                    00117 ;       EWEN routine
                    00118 ;
                    00119 ;*****************************************************
                    00120
0009                00121 EWEN
0009 1283           00122         bcf     STATUS, RP0     ; need to set bank 0
000A 1407           00123         bsf     PORTC, cs       ; set chip select line high
000B 3009           00124         movlw   b'00001001'     ; start bit is bit 3, 00 is
                    00125                                 ;  op code for EWEN
000C 2001           00126         call    output          ;
                    00127
000D 3080           00128         movlw   b'10000000'     ; 1 req for EWEN, 0000000 are don't cares
                    00129                                 ;  which is the 8 lsb address bits
000E 2001           00130         call    output          ;
                    00131
000F 1007           00132         bcf     PORTC, cs       ; bring chip select low to begin
                    00133                                 ;  EEPROMs internal write cycle
0010 3400           00134         retlw   0               ; return from subroutine
                    00135
                    00136 ; remember CS must be low for at least 250 nsec
                    00137 ;*****************************************************
                    00138
                    00139 ;*****************************************************
                    00140 ;
                    00141 ;       This routine outputs the two bytes required to send
                    00142 ;       the start bit, op code bits, and address bits
                    00143 ;
                    00144 ;*****************************************************
0011                00145 WRITE
0011 1283           00146         bcf     STATUS, RP0     ; need to set bank 0
0012 1407           00147         bsf     PORTC, cs       ; set chip select line high
0013 0829           00148         movf    hibyte, 0       ; put hibyte in w reg
                    00149
0014 2001           00150         call    output          ;
                    00151
0015 0804           00152         movf    FSR, 0          ; put addr pointed to by FSR into
                    00153                                 ;  w reg
0016 2001           00154         call    output          ;
                    00155
0017 082B           00156         movf    datbyt, 0       ; get ready to output data in datbyt
0018 2001           00157         call    output          ;
                    00158
0019 1007           00159         bcf     PORTC, cs       ; bring chip select low to begin
                    00160                                 ;  EEPROMs internal write cycle
                    00161
001A 0A84           00162         incf    FSR, 1          ; point to next location to
                    00163                                 ;  write to
001B 3400           00164         retlw   0
                    00165 ;*****************************************************
                    00166
                    00167 ;*****************************************************
                    00168 ;
                    00169 ;       This is the module that reads one byte of data
                    00170 ;
                    00171 ;*****************************************************
                    00172
001C                00173 READ
001C 1283           00174         bcf     STATUS, RP0     ; need to set bank 0
001D 1407           00175         bsf     PORTC, cs       ; set chip select line high
001E 1614           00176         bsf     SSPCON, CKP     ; make sure CKP is 1 to output
                    00177                                 ; next intstruction and addr
                    00178
                    00179
001F 0829           00180         movf    hibyte, 0       ; move data from hibyte to w
0020 2001           00181         call    output          ;
                    00182
```


© 1997 Microchip Technology Inc.

```
0021 082A           00183         movf    lobyte, 0     ; get ready to send next byte
                    00184                               ; which is the 8 lsb address bits
0022 2001           00185         call    output        ;
                    00186
                    00187 ;*****************************************************
                    00188 ;       This is where CKP bit is reset for receiving data.
                    00189 ;*****************************************************
0023 1214           00190         bcf     SSPCON, CKP   ; change clock polarity to 0
                    00191
0024 3000           00192         movlw   0x00          ; The byte xmitted here is a
                    00193                               ; don't care
0025 2001           00194         call    output        ;
                    00195
0026 1007           00196         bcf     PORTC, cs     ; bring chip select low to
                    00197                               ; terminate read command
                    00198
0027 0180           00199         clrf    INDF          ; clr location pointed to by FSR
0028 0824           00200         movf    rxdata, 0     ; move received data to w reg
0029 0080           00201         movwf   INDF          ; put received data in location
                    00202                               ; pointed to by FSR
                    00203
002A 0A84           00204         incf    FSR, 1        ; point to next location to
                    00205                               ; write to
002B 0AAA           00206         incf    lobyte, 1     ; next addr to read from
                    00207
002C 3400           00208         retlw   0               ;
                    00209 ;*************************************************************
                    00210
                    00211 ;*************************************************************
                    00212 ;*************************************************************
                    00213 ;*************************************************************
002D                00214 Start
002D 1283           00215         bcf     STATUS, RP0   ; need to set bank 0
002E 0187           00216         clrf    PORTC         ; initialize port c
002F 1683           00217         bsf     STATUS, RP0   ; need to set bank 1
0030 3010           00218         movlw   0x10          ; all bits are outputs except SDI
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0031 0087           00219         movwf   TRISC         ;  for SPI input
                    00220
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0032 018C           00221         clrf    PIE1          ; disables all peripheral ints
0033 018B           00222         clrf    INTCON        ; disables all interrupts
                    00223
0034 1283           00224         bcf     STATUS, RP0   ; need to set bank 0
0035 0194           00225         clrf    SSPCON        ; clear SSP control register
0036 3031           00226         movlw   0x31          ; SPI master, clk/16, ckp=1
0037 0094           00227         movwf   SSPCON        ;  SSPEN enabled
                    00228
0038 2009           00229         call    EWEN          ; output EWEN for enabling writes
                    00230
                    00231 ;*********************************************************
                    00232 ;       The next thing we will do is to write 0x5A to locations
                    00233 ;       10h through 13h.
                    00234 ;*********************************************************
                    00235
0039 300A           00236         movlw   b'00001010'   ; start bit is bit 3, 01 is
                    00237                               ;  op code for write
003A 00A9           00238         movwf   hibyte        ; load into hibyte
                    00239
003B 3010           00240         movlw   0x10          ; put beginning address in FSR
003C 0084           00241         movwf   FSR           ; for later use
                    00242
003D 305A           00243         movlw   b'01011010'   ; load 0x5A as data to be sent out
003E 00AB           00244         movwf   datbyt        ;
                    00245
003F 2011           00246 wrnext  call    WRITE         ; call write subroutine
```


© 1997 Microchip Technology Inc.

```
                    00247
                    00248 ;*******************************************************
                    00249 ;       Ready/Busy poll to decide when write is complete
                    00250 ;       and the 93LC66 is available for writing the next
                    00251 ;       byte.
                    00252 ;*******************************************************
                    00253
0040 0000           00254         nop                     ; cs must be low for > 250 ns
0041 1407           00255         bsf     PORTC, cs       ;  and then be brought high
                    00256
0042 1E07           00257 rbusy   btfss   PORTC, sdi      ; test ready/busy status
                    00258                                 ; if 1, internal write is done
0043 2842           00259         goto    rbusy           ; part still writing, stay in
                    00260                                 ;  loop
0044 1007           00261         bcf     PORTC, cs       ; bring cs back low
                    00262
0045 1D04           00263         btfss   FSR, 2          ; have we written all 4 locations?
0046 283F           00264         goto    wrnext          ; no, then write next byte
                    00265
                    00266 ;*****************************************************
                    00267 ;       Now, lets read back 10h through 13h (non-sequentially) and
                    00268 ;       store it in ram locations 20h through 23h in the 16C74.
                    00269 ;       With the Picmaster, I can read those memory locations
                    00270 ;       and see that it was read in properly.  This is how
                    00271 ;       I can quickly verify that the read function is working
                    00272 ;       properly.
                    00273 ;*****************************************************
                    00274
0047 3020           00275         movlw   0x20            ; where in RAM to begin storing
0048 0084           00276         movwf   FSR             ;  data read back from EEPROM
                    00277
0049 3010           00278         movlw   0x10            ; addr of where to begin reading
004A 00AA           00279         movwf   lobyte          ;  EEPROM from
                    00280
004B 300C           00281         movlw   b'00001100'     ; start bit is bit 3, 10 is
004C 00A9           00282         movwf   hibyte          ;  op code for read
                    00283
004D 201C           00284 rdnext  call    READ            ;
                    00285
004E 1D04           00286         btfss   FSR, 2          ; have we 20h thru 23?
004F 284D           00287         goto    rdnext          ; no, then read next location
                    00288
                    00289 ;*****************************************************
                    00290 ;       While program is in limbo routine, it is possible
                    00291 ;       to halt the processor with Picmaster and look at
                    00292 ;       the data contained 16C74 RAM locations 20h through 23h.
                    00293 ;*****************************************************
0050 0000           00294 limbo   nop
0051 2850           00295         goto    limbo
                    00296 ;
                    00297 ;
                    00298         end
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XX-------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:    82
Program Memory Words Free:  4014


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     3 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX B: MOTOROLA 68HC11 SOURCE CODE

```
;*************************************************************************
; 3-Wire byte write and byte read using SPI port (220 bytes)
;
; This program (hc11sp66.*) writes 4 bytes of $5A to a Microchip 93LC66
; using the SPI port of the Motorola 68HC11 microcontroller.  Ready/busy
; polling is used to determine when the current byte is done writing and
; the next byte is ready to be written to the 93LC66.  After the 4 bytes
; are written, they are read back and stored in RAM locations $100-$103.
; The evaluation board can be stopped after receiving the 4 bytes to
; view RAM locations and verify that what was written to the 93LC66 is
; what is read back.
;
; This code was written by Keith Pazul on 3/7/95.
;
; This is the way the bytes will be aligned to be sent to the 93LC56/66:
;
; First byte (where start bit, op codes reside)
;
; |  0  |  0  |  0  |  0  | SB  | OP1 | OP0 | A8  |
;
; where SB is start bit, OP1 is op code msb, OP0 is op code lsb,
; and A8 is address msb.
;
; Next byte (address)
;
; | A7  | A6  | A5  | A4  | A3  | A2  | A1  | A0  |
;
; where A8-A0 are address bits required to address 4k bits of memory.
;
; This code will work for any 93 series device.  The only difference
; is that the number of address bits is adjusted and the start bit
; and op code bits are adjusted to follow directly after the address
; MSB.
;
; THE 93LC56 or 66 IS ASSUMED TO BE IN x8 MODE.  IT REQUIRES
; THE USER TO TIE THE ORG PIN TO Vss.
;
; This program was written using a 68CH11EVBU evaluation board.
; This board has a monitor program in firmware of the 68HC11 which
; allows single stepping, register viewing, modifying, etc.  Since this
; is the case, the program code will be loaded into the on-chip EEPROM.
; This EEPROM begins at $B600.  The control registers are left to thier
; default location of $1000 and the RAM is left to its default location.
; RAM locations $48-$ff are used by the monitor program and are not
; available for program use.  Therefore, the stack pointer is set a $47
; and will be able to use all of the RAM to $00, and the RAM variables
; begin at location $100 and go up from there.  I cannot program the
; reset vector since it is ROM space (at $FFFE), so the way I run this
; program is to use the monitor program that comes with the evaluation
; board to set the program counter to the starting address of my user
; program ($B600) and begin from there.  For users who do have access
; to the reset vector, the label of the beginning program (in my case, it
; is called START) should be loaded at location $FFFE.  This program was
; not assembled using a Motorola assembler, but was assembled using
; Universal Cross-Assemblers Cross-32 Meta-Assembler.  It has the
; ability to assemble just about any microcontroller code.  There are
; certain commands that are unique to the cross-assembler.  These
; commands will be commented differently than other comments to
; be recognizable.  They will look like this:
```


© 1997 Microchip Technology Inc.

```
;*********************************
; Special cross-assembler command(s)
;*********************************

; I do not know the exact assembler commands required to accomplish
; assembly using a Motorola assembler.
;
; The crystal that comes with the evaluation board is 8 Mhz.
;
; The SPI port in on port D.  The bits are defined as follows:
;
;           MISO        PORT D, PIN 2 (pin 22 on package)
;           MOSI        PORT D, PIN 3 (pin 23 on package)
;           SCK         PORT D, PIN 4 (pin 24 on package)
;           SS\         PORT D, PIN 5 (pin 25 on package)
;           CS          PORT C, PIN 6
;
; Note that only the CS pin resides on port C.
;*************************************************


;************************************************
        CPU     "C:\WINC32\68HC11.TBL"    ; LOAD TABLE
        HOF     "MOT8"; Hex output is Motorola S-records
;************************************************
;
;****************************************************************
; 68HC11 control register locations
;****************************************************************
REGBS       EQU         1000H               ; BEGINNING OF REGISTERS
RAMBS       EQU         100H                ; BEGINNING OF RAM VARIABLES
DDRC        EQU         REGBS+07H           ; DATA DIRECTION REG FOR PORT C
PORTC       EQU         REGBS+03H           ; PORT C DATA REGISTER
PCOFF       EQU         03H                 ; OFFSET FROM CONTROL REG BEG.
PORTD       EQU         1008H               ; PORT D DATA REGISTER
DDRD        EQU         1009H               ; PORT D DATA DIRECTION REGISTER
SPCR        EQU         1028H               ; SPI CONTROL REGISTER
SPSR        EQU         1029H               ; SPI STATUS REGISTER
SPDR        EQU         102AH               ; SPE DATA REGISTER
;****************************************************************
;****************************************************************
; User defined constants
;****************************************************************
CSMASK      EQU         01000000B           ; BIT MASK FOR CHIP SELECT
SDIMASK     EQU         00000100B           ; BIT MASK FOR MISO PIN
;****************************************************************

            LDS         #0047H              ; STACK POINTER BEGINS AT $47

            ORG         100H                ; RAM variables begin at 100h

;************************************************
; DFS is a Universal Cross-Assembler directive that stands for define
; storage.  1 is for byte, 2 is for word, 4 is for long word.  These are
; the user defined RAM variables.

RXARAY      DFS         1 * {4}             ; 100H-103H
RXDATA      DFS         1                   ; 104H
DATBYT      DFS         1                   ; 105H
HIBYTE      DFS         1                   ; 106H
LOBYTE      DFS         1                   ; 107H
RBTEST      DFS         1                   ; 108H
ADDROFF     DFS         1                   ; 109H
;************************************************

;****************************************************************
```


© 1997 Microchip Technology Inc.

```
; Program code cannot be placed in ROM for eval board because
; eval board firmware is loaded in ROM.  Program code will be
; loaded in EEPROM (which is 512 bytes and begins at B600h).
;
          ORG     0B600H
;*************************************************************
;
; This is the main portion of the code.  It is where the reset
; vector should set program counter to.
;
;*************************************************************

START
          LDX       #REGBS          ; LOADS BEG OF REGISTERS INTO X

          BCLR      PCOFF,X,CSMASK  ; MASK SURE CS IS CLEARED

          LDAA      #10110000B      ; SETS UP BITS 0-3 AND 6 AS INPUTS,
          STAA      DDRC            ;  BITS 4,5, AND 7 AS OUTPUTS

          CLR       PORTC           ; CLEAR ALL PORT C BITS

          LDAA      #11111011B      ; ALL BITS ARE OUTPUTS EXCEPT MISO
          STAA      DDRD            ;

          LDAA      #01010010B      ; SPIE=0,SPE=1,DWOM=0,MSTR=1,
          STAA      SPCR            ;  CPOL=0,CPHA=0, CLK/16

          LDAA      #SDIMASK        ; STORE READY/BUSY MASK IN
          STAA      RBTEST          ;  LOCATION RBTEST

          LDAA      #10H            ;
          STAA      LOBYTE          ; LOAD 8 LSB'S OF ADDRESS

          CLR       ADDROFF         ; SET ADDRESS OFFSET TO 0 FOR
                                    ;  READ COMMANDS

          JSR       EWEN            ; SEND EWEN COMMAND

;*********************************************************
; Now lets write 5Ah out to address 10h
;*********************************************************

          LDAA      #00001010B      ; STRT BIT IS BIT 3, 01 IS
          STAA      HIBYTE          ;  OP CODE FOR WRITE, LSB IS
                                    ;  ADDRESS

          LDAA      #01011010B      ; LOAD 0x5A IN DATA LOCATION
          STAA      DATBYT          ;

WRNEXT    JSR       WRITE           ; CALL WRITE ROUTINE

;*************************************************************
;   ready/busy poll input pin to see when internal write
;   cycle is complete.
;*************************************************************

          BSET      PCOFF,X,CSMASK  ; RAISE CS FOR READY/BUSY POLLING

RBUSY     LDAA      PORTD           ; INPUT PORT D
          ANDA      RBTEST          ; MASK OFF ALL BITS EXCEPT MISO
          BEQ       RBUSY           ; IF MISO IS LO, PART STILL BUSY,
                                    ;  ELSE WRITE IS COMPLETE

          BCLR      PCOFF,X,CSMASK  ; POLLING COMPLETE, END CHIP SELECT
```


© 1997 Microchip Technology Inc.

```
            LDAA        LOBYTE              ; GET ADDRESS OFFSET
            BITA        #00000100B          ; ARE ALL BYTES WRITTEN YET?
            BEQ         WRNEXT              ; ALL BYTES HAVE NOT BEEN WRITTEN,
                                            ;  WRITE NEXT BYTE

;*****************************************************************
;   Now lets read back the location of rxdata to see if we
;   read back what we wrote to that location.
;*****************************************************************

            LDAA        #00001100B          ; STRT BIT IS BIT 3, 10 IS
            STAA        HIBYTE              ;  OP CODE FOR READ, LSB IS
                                            ;  ADDRESS
            LDAA        #10H                ; RESET BEGINNING ADDRESS
            STAA        LOBYTE              ;  FOR READ OPERATIONS

            LDY         #RAMBS              ; RX BUFF IS AT BEG OF RAM

RDNEXT      JSR         READ                ; CALL READ SUBROUTINE

            LDAA        LOBYTE              ; LOAD ACC WITH MASK THAT
                                            ;  CHECKS TO SEE IF 4 BYTES
                                            ;  HAVE BEEN WRITTEN
            BITA        #00000100B          ; CHECK TO SEE IF 4 BYTES HAVE
            BEQ         RDNEXT              ;  BEEN WRITTEN.  IF NOT, READ
                                            ;  NEXT BYTE.

            LDAA        #10110000B          ; DISABLES CS TO PROHIBIT
            STAA        DDRC                ;  POSSIBILITY OF INADVERTANT
                                            ;  WRITES

LIMBO       NOP
            JMP         LIMBO

;*****************************************************************
;
; This subroutine outputs the data stored in TXBUFF
;
;*****************************************************************

OUTPUT
            STAA        SPDR                ; MOVE DATA FROM ACC A TO SPDR
LOOP1       LDAB        SPSR                ; SEE IF XFER COMPLETE FLAG SET
            BPL         LOOP1               ; IF NOT, LOOP UNTIL SET

            LDAA        SPDR                ; MOVES RECEIVED BYTE FROM DATA REG
            STAA        RXDATA              ;  TO RECEIVE DATA LOCATION

            RTS
;*****************************************************************

;*****************************************************************
;
;           EWEN routine
;
;*****************************************************************
EWEN
            LDAA        #11110000B          ; SET CS FROM IN TO OUT.  ALL
            STAA        DDRC                ;  OTHER BITS ARE THE SAME

            BSET        PCOFF,X,CSMASK      ; DROP CS TO BEGIN COMMAND

            LDAA        #00001001B          ; STRT BIT IS BIT 3, 00 IS
                                            ;  OP CODE FOR EWEN
            JSR         OUTPUT              ;
```


© 1997 Microchip Technology Inc.

```
            LDAA        #10000000B          ; 1 IS REQ FOR EWEN.  REST
                                            ;  ARE DON'T CARE BITS FOR
                                            ;  LOWER 8 ADDR BITS
            JSR         OUTPUT              ;

            BCLR        PCOFF,X,CSMASK      ; SET CS TO COMPLETE EWEN

            RTS
;*************************************************************

;*************************************************************
;
;           WRITE routine
;
;*************************************************************
WRITE
            BSET        PCOFF,X,CSMASK      ; BEGIN WRITE COMMAND

            LDAA        HIBYTE              ; GET DATA FROM HIBYTE TO OUTPUT
            JSR         OUTPUT              ;

            LDAA        LOBYTE              ; 8 LSBs OF ADDRESS POINTED TO
            JSR         OUTPUT              ;  BY ADDROFF OFFSET TO Y REG

            LDAA        DATBYT              ; OUTPUT DATA BYTE
            JSR         OUTPUT              ;

            BCLR        PCOFF,X,CSMASK      ; DROP CS TO BEGIN COMMAND

            INC         LOBYTE              ; INC ADDRESS FOR NEXT LOCATION
                                            ;  TO WRITE TO
            RTS
;*************************************************************

;*************************************************************
;
;           READ routine
;
;*************************************************************
READ
            LDAA        SPCR                ; READ IN SPI CONTROL REG
            ANDA        #11111011B          ; SET CPHA TO 0 FOR CONTROL
            STAA        SPCR                ;  PORTION OF READ SEQUENCE

            BSET        PCOFF,X,CSMASK      ; DROP CS TO BEGIN COMMAND

            LDAA        HIBYTE              ; OUTPUT HI BYTE FOR READ CMD
            JSR         OUTPUT              ;

            LDAA        LOBYTE              ; 8 LSBs OF ADDRESS
            JSR         OUTPUT              ;

;*************************************************************
;*************************************************************
;*************************************************************
; To read a byte, a byte must be transmitted.  Here we xmit a byte
; of don't care bits to receive a byte.  WE MUST CHANGE THE CPHA
; FROM 0 TO 1 TO CHANGE BEFORE DATA IS RECEIVED.  IF NOT, THEN THE
; MICRO WILL TRY TO RECEIVE AT THE SAME TIME THE 93XX66 WILL BE
; TRANSMITTING.  THE SYMPTOM OF THIS NOT SET UP PROPERLY IS THAT
; THE DATA READ WILL BE SHIFTED RIGHT ONE BIT WITH MSB BEING 0.
;*************************************************************
;*************************************************************
;*************************************************************
```


© 1997 Microchip Technology Inc.

```
          LDAA        SPCR              ; READ IN SPI CONTROL REG
          ORAA        #00000100B        ; SET CPHA TO 1
          STAA        SPCR

          JSR         OUTPUT            ; XMIT A BYTE OF DON'T CARE BITS
                                        ;  TO RECEIVE A BYTE

          BCLR        PCOFF,X,CSMASK    ; CLR CS TO END COMMAND

          LDAA        RXDATA            ; GET DATA RECIEVED IN RXDATA
          STAA        0, Y              ;  AND STORE IN LOCATION POINTED
                                        ;  TO BY Y INDEX REG + 0 OFFSET
          INY
          INC         LOBYTE            ; INC ADDR OF NEXT BYTE TO READ
                                        ;  FROM
          RTS
;*************************************************************

;+++++++++++++++++++++++++++++++++++++++++++++++++++
; This command tells the cross-assembler that code starts
; at the location of START

          END         START
;+++++++++++++++++++++++++++++++++++++++++++++++++++
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN614

Interfacing the 8051 with 2-wire Serial EEPROMs

Author: *Mike Rosenfield*
Microchip Technology Inc.

INTERFACING MICROCHIP SERIAL EEPROMS TO THE INTEL 8051 FAMILY OF MICROCONTROLLERS

Many designers today are implementing embedded systems that require low cost non-volatile memory. Microchip has addressed this need with a full line of serial EEPROMs, in a variety of memory configurations, using the industry-standard 2- or 3-wire communication protocols. The theory and application of these protocols are addressed in detail in Microchip's application note AN536.

Microchip recognizes that its broad customer base uses a variety of micro-controllers; many firmware related questions have been asked concerning interfacing the 8051 family and its derivatives.

The purpose of this app note is to provide assembly language examples of 8051 code for the various serial EEPROMs available from Microchip. These routines are intended to provide the basic operating kernels for storing data to or retrieving data from a serial EEPROM.

All of the routines in this app note are available, as source code, for downloading from Microchip's BBS. Information on the BBS is available elsewhere in the Embedded Control Handbook. The file to download is AN614213.zip.

This app note covers all of Microchip's 2-wire serial devices. Note that some devices have features not supported in others, and therefore, some sections of the code presented here may not be applicable to a particular part. We have attempted to label those special sections to minimize confusion.

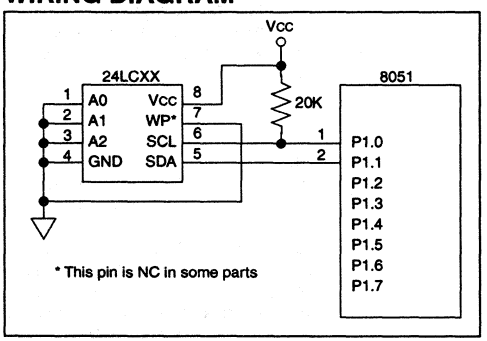
The code includes a simple loop-type shell to enable it to be executed (with an emulator) without the user having to write any other routines. The various address and data pointers must be set to the desired values by hand, before each execution cycle.

TIMING DATA

Clock and data timing is accomplished by software. There are two sets of timing specifications: 100 kHz and 400 kHz. Assuming a 12 MHz 8051 clock, extra NOP's have been added to slow timing down to 100 kHz. See Note 1 in the listing. If a 16 MHz clock is used, additional NOP's are required for 100 kHz operation. See Note 2 in the listing. For 400 kHz operation, the NOP's labelled Note 1 or Note 2 are not needed. If not needed, NOP's may be left out.

Below is the connection diagram used for this app note. Do not forget the pull-up resistor!.

WIRING DIAGRAM



7

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code.
Microchip's Worldwide Web Address: www.microchip.com;
Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: SOURCE CODE

```
MCS-51 MACRO ASSEMBLER
2WIRE                                                          05/31/95

DOS 6.0 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN 2WIRE.OBJ
ASSEMBLER INVOKED BY:  C:\ICE5100\ASM51\ASM51.EXE 2WIRE.TXT


LOC  OBJ       LINE    SOURCE

               1       $ PAGELENGTH(46) PAGEWIDTH(132) DEBUG NOPAGING XREF
               2       ;REGISTER ASSIGNMENTS:
               3   ;
               4   ;R1      DATA OR DATA POINTER
               5   ;R2      LOOP COUNTER REGISTER
               6   ;R3      ADDRESS, HI BYTE
               7   ;R4      ADDRESS, LOW BYTE
               8   ;R5
               9   ;R6      BYTE COUNT FOR PAGE OPERATIONS
              10
              11   ;PIN ASSIGNMENTS:
              12   ;Port 1 bit 0 is data
              13   ;Port 1 bit 1 is clock
              14   ;
              15   ;       These routines assume chip address = 0
              16   ;
              17   ;       The oscillator frequency assumed for this app note is 12 MHz.
              18   ;
              19   ;       These routines use software timing loops. They may have to be
              20    ;      adjusted if a different oscillator frequency is used.
              21   ;
              22   ;NOTE 1 These NOP'S added for timing delays only on 'C' parts, OR 'LC' parts
              23   ;       where Vcc is less than 4.5 V. and the oscillator frequency is 12 MHz.
              24   ;       This allows a bit rate of 100kHz.
              25   ;NOTE 2 Use these NOP's with a 16 MHz oscillator and 100kHz bit rate.
              26   ;       For 400kHz bit rate, the NOP's in Note 1 and Note 2 are not required.
              27   ;
              28   ; The EEPROM will be busy after a write cycle is initiated (by a stop condition)
              29   ; for between 1mS to 10 mS per page (or per byte if a byte write). This app note
              30   ; assumes the user will program appropriate wait times after a write, or check
              31   ; for Busy status. A subroutine is provided to check the Busy Status.
              32   ;
              33   ;RAM DEFINITIONS
              34   ;
0030          35   ORG     30H
0030          36   BYTSTR: DS      20H             ;STORAGE FOR READ DATA
              37   ;
              38   ;CONSTANTS -- REDEFINE AS NECESSARY
              39   ;
  00A0        40   WTCMD   EQU     10100000B       ;WRITE DATA COMMAND Note 3
  00A1        41   RDCMD   EQU     10100001B       ;READ DATA COMMAND  Note 3
  0040        42   RDEND   EQU     01000000B       ;READ HIGH-ENDURANCE BLOCK NUMBER COMMAND
  0000        43   ADDRH   EQU     0
  0000        44   ADDRL   EQU     0
  0055        45   DTA     EQU     55H
  0008        46   BYTCNT  EQU     8
              47   ;
              48   ;Note3 Some chip or byte address bits are embedded in the control byte. Refer to
              49   ;      the data sheet for exact configuration, which varies from part to part.
              50   ;
```


© 1997 Microchip Technology Inc.

```
                51  ;*******************************************************************************
                52  ; This section contains test loop routines. They form a simple operating shell
                53  ; to allow the 2-wire interface code to be tested in a stand-alone mode. Using
                54  ; an emulator, change  "NONE" to one of the four listed routines
                55  ; to test that funcion. The address and data constants can also be set as
                56  ; desired.
                57  ; If using a 24XX32 or 24XX65, change the called routines by adding 'L' to the
                58  ; end of the name. This is required because these parts use TWO address bytes.
                59  ; The 'L' routines send out the extra address byte.
                60  ;*******************************************************************************
0000            61  ORG     0
0000 020003     62              JMP     START
                63
0003 7590FF     64  START:  MOV     P1,#0FFH        ;INIT PORT 1
0006 12000B     65  CALL    NONE            ;TEST LOOP INSERT PROPER ADDRESS HERE
0009 80F8       66  JMP     START
                67
000B 22         68     NONE:   RET
                69  ;*
                70  ;* WRITE ONE BYTE TO EEPROM
                71  ;* The Address Pointer is the address in the EEPROM. The byte to be sent to the
                72  ;* EEPROM is stored in the constant 'DTA'
                73  ;*
000C 7B00       74  TESTWR: MOV     R3,#ADDRH       ;LOAD ADDRESS POINTER FOR 24XX32 OR 24XX65 ONLY
000E 7C00       75          MOV     R4,#ADDRL       ;LOAD ADDRESS POINTER FOR ALL DEVICES
0010 7955       76          MOV     R1,#DTA         ;LOAD DATA BYTE
0012 120037     77          CALL    BYTEW           ;CALL  BYTE WRITE ROUTINE
0015 22         78          RET
                79
                80  ;*
                81  ;* WRITE A BLOCK OF DATA TO EEPROM
                82  ;* The address pointer is the address in EEPROM where data will start. The byte
                83  ;* pointer is the starting address of RAM containing the block of data to be
                84  ;* sent. The byte count indicates how many bytes to send to the EEPROM.
                85  ;* The number of bytes that can be sent before a STOP command is issued is
                86  ;* dependant on EEPROM type. Refer to the data book for specific values.
                87  ;*
0016 7B00       88  BLKWR:  MOV     R3,#ADDRH       ;LOAD ADDRESS POINTER FOR 24XX32 OR 24XX65 ONLY
0018 7C00       89          MOV     R4,#ADDRL       ;LOAD ADDRESS POINTER FOR ALL DEVICES
001A 7930       90          MOV     R1,# BYTSTR     ;LOAD BYTE POINTER
001C 7E08       91          MOV     R6,#BYTCNT      ;LOAD BYTE COUNT
001E 120048     92          CALL    PAGEW           ;CALL PAGE WRITE ROUTINE
0021 22         93          RET
                94
                95  ;*
                96  ;* READ ONE BYTE FROM EEPROM
                97  ;* The address pointer is the address of the desired byte in EEPROM.
                98  ;* The byte will be returned in R1.
                99  ;*
0022 7B00      100 TESTRD: MOV     R3,#ADDRH       ;LOAD ADDRESS POINTER FOR 24XX32 OR 24XX65 ONLY
0024 7C00      101         MOV     R4,#ADDRL       ;LOAD ADDRESS POINTER FOR ALL DEVICES
0026 120082    102         CALL    BYTERD          ;CALL BYTE READ ROUTINE.
0029 F9        103         MOV     R1,A            ;SAVE THE BYTE
002A 22        104         RET
               105
               106 ;*
               107 ;* READ A BLOCK FROM EEPROM
               108 ;* The address pointer is the starting address of the desired data block in
               109 ;* EEPROM. The data pointer is the starting address in RAM where data will be
               110 ;* stored. The byte count indicates how many bytes should be read.
               111 ;* The entire EEPROM may be read with one READ command this way.
               112 ;*
002B 7B00      113 BLOKRD: MOV     R3,#ADDRH       ;LOAD ADDRESS POINTER FOR 24XX32 OR 24XX65 ONLY
002D 7C00      114         MOV     R4,#ADDRL       ;LOAD ADDRESS POINTER FOR ALL DEVICES
002F 7930      115         MOV     R1,#BYTSTR      ;LOAD DATA POINTER
0031 7E08      116         MOV     R6,#BYTCNT      ;LOAD BYTE COUNT
```


© 1997 Microchip Technology Inc.

```
0033 12005C    117          CALL    BLKRD           ;CALL BLOCK READ ROUTINE
0036 22        118          RET
               119
               120 ;*END OF TEST LOOP
               121 ;*****************************************************************************
               122
               123
               124 ;*********************************************************************
               125 ; This routine writes a byte of data to EEPROM
               126 ; The EEPROM address is assumed to be in R4. See NOTE 3.
               127 ; The DATA to be written is assumed to be in R1
               128 ;*********************************************************************
0037 74A0      129     BYTEW:  MOV     A,#WTCMD        ;LOAD WRITE COMMAND
0039 120120    130             CALL    OUTS            ;SEND IT
003C EC        131             MOV     A,R4            ;GET BYTE ADDRESS
003D 120158    132             CALL    OUT             ;SEND IT
0040 E9        133             MOV     A,R1            ;GET DATA
0041 120158    134             CALL    OUT             ;SEND IT
0044 120197    135             CALL    STOP            ;SEND STOP CONDITION
0047 22        136             RET
               137
               138 ;*****************************************************************************
               139 ; THIS ROUTINE WRITES A PAGE OF DATA TO EEPROM
               140 ; The EEPROM start address is assumed to be in R4. See NOTE 3.
               141 ; The DATA pointer is in R1
               142 ; The BYTE count is in R6
               143 ; The number of bytes that can be transfered depends on the EEPROM used.
               144 ;*****************************************************************************
0048 74A0      145     PAGEW:  MOV     A,#WTCMD        ;LOAD WRITE COMMAND
004A 120120    146             CALL    OUTS            ;SEND IT
004D EC        147             MOV     A,R4            ;GET LOW BYTE ADDRESS
004E 120158    148             CALL    OUT             ;SEND IT
0051 E7        149     BTLP:   MOV     A,@R1           ;GET DATA
0052 120158    150             CALL    OUT             ;SEND IT
0055 09        151             INC     R1              ;INCREMENT DATA POINTER
0056 DEF9      152             DJNZ    R6,BTLP         ;LOOP TILL DONE
0058 120197    153             CALL    STOP            ;SEND STOP CONDITION
005B 22        154             RET
               155
               156
               157 ;*****************************************************************************
               158 ; THIS ROUTINE READS A BLOCK OF DATA FROM EEPROM AT A SPECIFIED ADDRESS
               159 ; EEPROM address in R4. See NOTE 3.
               160 ; Stores data at RAM location pointed to by R1
               161 ; Byte count specified in R6. May be 1 to 256 bytes
               162 ;*****************************************************************************
005C 74A0      163     BLKRD:  MOV     A,#WTCMD        ;LOAD WRITE COMMMAND TO SEND ADDRESS
005E 120120    164             CALL    OUTS            ;SEND IT
0061 EC        165             MOV     A,R4            ;GET LOW BYTE ADDRESS
0062 120158    166             CALL    OUT             ;SEND IT
0065 74A1      167             MOV     A,#RDCMD        ;LOAD READ COMMMAND
0067 120120    168             CALL    OUTS            ;SEND IT
006A 12017E    169     BRDLP:  CALL    IN              ;READ DATA
006D F7        170             MOV     @R1,A           ;STORE DATA
006E 09        171             INC     R1              ;INCREMENT DATA POINTER
006F DE04      172             DJNZ    R6,AKLP         ;DECREMENT LOOP COUNTER
0071 120197    173             CALL    STOP            ;IF DONE, ISSUE STOP CONDITION
0074 22        174             RET                     ;DONE, EXIT ROUTINE
               175
0075 C290      176     AKLP:   CLR     P1.0            ;NOT DONE, ISSUE ACK
0077 D291      177     SETB    P1.1
0079 00        178             NOP                     ;NOTE 1
007A 00        179             NOP
007B 00        180             NOP
007C 00        181             NOP                     ;NOTE 2
007D 00        182             NOP
```


© 1997 Microchip Technology Inc.

```
007E C291       183             CLR     P1.1
0080 80E8       184             JMP     BRDLP           ;CONTINUE WITH READS
                185
                186     ;*********************************************************************
                187     ; THIS ROUTINE READS A BYTE OF DATA FROM THE EEPROM
                188     ; The EEPROM address is in R4. See NOTE 3.
                189     ; Returns the data byte in R1
                190     ;*********************************************************************
0082 74A0       191     BYTERD: MOV     A,#WTCMD        ;LOAD WRITE COMMMAND TO SEND ADDRESS
0084 120120     192             CALL    OUTS            ;SEND IT
0087 EC         193             MOV     A,R4            ;GET LOW BYTE ADDRESS
0088 120158     194             CALL    OUT             ;SEND IT
008B 12008F     195             CALL    CREAD           ;GET DATA BYTE
008E 22         196             RET
                197
                198     ;*********************************************************************
                199     ; THIS ROUTINE READS A BYTE OF DATA FROM EEPROM
                200     ; From EEPROM current address pointer.
                201     ; Returns the data byte in R1
                202     ;*********************************************************************
008F 74A1       203     CREAD:  MOV     A,#RDCMD        ;LOAD READ COMMMAND
0091 120120     204             CALL    OUTS            ;SEND IT
0094 12017E     205             CALL    IN              ;READ DATA
0097 F9         206             MOV     R1,A            ;STORE DATA
0098 120197     207             CALL    STOP            ;SEND STOP CONDITION
009B 22         208             RET
                209
                210     ;*********************************************************************
                211     ; The next four routines are used with the 24XX32 or 24XX65 only. These
                212     ; parts require two address bytes, and these routines send the second byte
                213     ; out. Other than this, these routines are the same as the previous four.
                214     ;*********************************************************************
                215     ; THIS ROUTINE READS A BLOCK OF DATA FROM EEPROM AT A SPECIFIED ADDRESS
                216     ; This routine is for the 24LC32 or 24LC64
                217     ; EEPROM address in R3:R4
                218     ; Stores data at RAM location pointed to by R1
                219     ; Byte count specified in R6. May be 1 to 256 bytes
                220     ;*********************************************************************
009C 74A0       221     BLKRDL: MOV     A,#WTCMD        ;LOAD WRITE COMMMAND TO SEND ADDRESS
009E 120120     222             CALL    OUTS            ;SEND IT
00A1 EB         223             MOV     A,R3            ;GET HI BYTE ADDRESS
00A2 120158     224             CALL    OUT             ;SEND IT
00A5 EC         225             MOV     A,R4            ;GET LOW BYTE ADDRESS
00A6 120158     226             CALL    OUT             ;SEND IT
00A9 74A1       227             MOV     A,#RDCMD        ;LOAD READ COMMMAND
00AB 120120     228             CALL    OUTS            ;SEND IT
00AE 80BA       229             JMP     BRDLP           ;CONTINUE WITH DATA READ
                230
                231     ;*********************************************************************
                232     ; This routine writes a byte of data to EEPROM
                233     ; This routine is for the 24LC32 or 24LC64
                234     ; The EEPROM address is assumed to be in R3:R4
                235     ; The DATA to be written is assumed to be in R1
                236     ;*********************************************************************
00B0 74A0       237     BYTEWL: MOV     A,#WTCMD        ;LOAD WRITE COMMAND
00B2 120120     238             CALL    OUTS            ;SEND IT
00B5 EB         239             MOV     A,R3            ;GET HI BYTE ADDRESS
00B6 120158     240             CALL    OUT             ;SEND IT
00B9 EC         241             MOV     A,R4            ;GET LOW BYTE ADDRESS
00BA 120158     242             CALL    OUT             ;SEND IT
00BD E9         243             MOV     A,R1            ;GET DATA
00BE 120158     244             CALL    OUT             ;SEND IT
00C1 120197     245             CALL    STOP            ;SEND STOP CONDITION
00C4 22         246             RET
                247
                248     ;*********************************************************************
```


© 1997 Microchip Technology Inc.

```
                    249     ; THIS ROUTINE WRITES A PAGE OF DATA TO EEPROM
                    250     ; This routine is for the 24LC32 or 24LC64
                    251     ; The EEPROM start address is assumed to be in R3:R4
                    252     ; The DATA pointer is in R1
                    253     ; The BYTE count is in R6
                    254     ; The number of bytes that can be transfered depends on the EEPROM in use.
                    255     ;*****************************************************************
00C5 74A0           256     PAGEWL: MOV     A,#WTCMD        ;LOAD WRITE COMMAND
00C7 120120         257             CALL    OUTS            ;SEND IT
00CA EB             258             MOV     A,R3            ;GET HI BYTE ADDRESS
00CB 120158         259             CALL    OUT             ;SEND IT
00CE EC             260             MOV     A,R4            ;GET LOW BYTE ADDRESS
00CF 120158         261             CALL    OUT             ;SEND IT
00D2 E7             262             BTLPL:  MOV     A,@R1           ;GET DATA
00D3 120158         263             CALL    OUT             ;SEND IT
00D6 09             264             INC     R1              ;INCREMENT DATA POINTER
00D7 DEF9           265             DJNZ    R6,BTLPL        ;LOOP TILL DONE
00D9 120197         266             CALL    STOP            ;SEND STOP CONDITION
00DC 22             267             RET
                    268
                    269     ;*****************************************************************
                    270     ; THIS ROUTINE READS A BYTE OF DATA FROM THE EEPROM
                    271     ; This routine is for the 24LC32 or 24LC64
                    272     ; The EEPROM address is in R3:R4
                    273     ; Returns the data byte in R1
                    274     ;*****************************************************************
00DD 74A0           275     BYTERDL:MOV     A,#WTCMD        ;LOAD WRITE COMMMAND TO SEND ADDRESS
00DF 120120         276             CALL    OUTS            ;SEND IT
00E2 EB             277             MOV     A,R3            ;GET HI BYTE ADDRESS
00E3 120158         278             CALL    OUT             ;SEND IT
00E6 EC             279             MOV     A,R4            ;GET LOW BYTE ADDRESS
00E7 120158         280             CALL    OUT             ;SEND IT
00EA 118F           281             CALL    CREAD           ;GET DATA BYTE
00EC 22             282             RET
                    283
                    284     ;
                    285     ;SUBROUTINES
                    286     ;
                    287     ;*****************************************************************
                    288     ; This routine tests for WRITE DONE condition
                    289     ; by testing for an ACK.
                    290     ; This routine can be run as soon as a STOP condition
                    291     ; has been generated after the last data byte has been
                    292     ; sent to the EEPROM. No ACK will be returned until
                    293     ; the EEPROM is done with the write operation.
                    294     ;*****************************************************************
00ED 74A0           295     ACKTST: MOV     A,#WTCMD        ;LOAD WRITE COMMMAND TO SEND ADDRESS
00EF 7A08           296             MOV     R2,#8           ;LOOP COUNT  -- EQUAL TO BIT COUNT
00F1 C290           297             CLR     P1.0            ;START CONDITION -- DATA = 0
00F3 00             298             NOP                     ;NOTE 1
00F4 00             299             NOP
00F5 00             300             NOP
00F6 00             301             NOP                     ;NOTE 2
00F7 00             302             NOP
00F8 C291           303             CLR     P1.1            ;CLOCK = 0
00FA 33             304             AKTLP:  RLC     A               ;SHIFT BIT
00FB 5005           305             JNC     AKTLS
00FD D290           306             SETB    P1.0            ;DATA = 1
00FF 020104         307             JMP     AKTL1           ;CONTINUE
0102 C290           308             AKTLS:  CLR     P1.0            ;DATA = 0
0104 D291           309             AKTL1:  SETB    P1.1            ;CLOCK HI
0106 00             310             NOP                     ;NOTE 1
```


© 1997 Microchip Technology Inc.

```
0107 00          311            NOP
0108 00          312            NOP
0109 00          313            NOP                     ;NOTE 2
010A 00          314            NOP
010B C291        315            CLR     P1.1            ;CLOCK LOW
010D DAEB        316            DJNZ    R2,AKTLP        ;DECREMENT COUNTER
010F D290        317            SETB    P1.0            ;TURN PIN INTO INPUT
0111 00          318            NOP                     ;NOTE 1
0112 00          319            NOP                     ;NOTE 2
0113 D291        320            SETB    P1.1            ;CLOCK ACK
0115 00          321            NOP                     ;NOTE 1
0116 00          322            NOP
0117 00          323            NOP
0118 00          324            NOP                     ;NOTE 2
0119 00          325            NOP
011A 309002      326            JNB     P1.0,EXIT       ;EXIT IF NO ACK (WRITE NOT DONE)
011D C291        327            CLR     P1.1
                 328    ;                               ;SET DONE FLAG
011F 22          329    EXIT:    RET
                 330
                 331
                 332
                 333    ;*************************************************************************
                 334    ; THIS ROUTINE SENDS OUT CONTENTS OF THE ACCUMULATOR
                 335    ; to the EEPROM and includes START condition. Refer to the data sheets
                 336    ;for discussion of START and STOP conditions.
                 337    ;*************************************************************************
                 338
0120 7A08        339    OUTS:   MOV     R2,#8           ;LOOP COUNT  -- EQUAL TO BIT COUNT
0122 D290        340            SETB    P1.0            ;INSURE DATA IS HI
0124 D291        341            SETB    P1.1            ;INSURE CLOCK IS HI
0126 00          342            NOP                     ;NOTE 1
0127 00          343            NOP
0128 00          344            NOP
0129 00          345            NOP                     ;NOTE 2
012A 00          346            NOP
012B C290        347            CLR     P1.0            ;START CONDITION -- DATA = 0
012D 00          348            NOP                     ;NOTE 1
012E 00          349            NOP
012F 00          350            NOP
0130 00          351            NOP                     ;NOTE 2
0131 00          352            NOP
0132 C291        353            CLR     P1.1            ;CLOCK = 0
0134 33          354            OTSLP:  RLC     A               ;SHIFT BIT
0135 5005        355            JNC     BITLS
0137 D290        356            SETB    P1.0            ;DATA = 1
0139 02013E      357            JMP     OTSL1           ;CONTINUE
013C C290        358            BITLS:  CLR     P1.0            ;DATA = 0
013E D291        359    OTSL1:   SETB    P1.1            ;CLOCK HI
0140 00          360            NOP                     ;NOTE 1
0141 00          361            NOP
0142 00          362            NOP
0143 00          363            NOP                     ;NOTE 2
0144 00          364            NOP
0145 C291        365            CLR     P1.1            ;CLOCK LOW
0147 DAEB        366            DJNZ    R2,OTSLP        ;DECREMENT COUNTER
0149 D290        367            SETB    P1.0            ;TURN PIN INTO INPUT
014B 00          368            NOP                     ;NOTE 1
014C 00          369            NOP                     ;NOTE 2
014D 00          370            NOP
014E D291        371            SETB    P1.1            ;CLOCK ACK
0150 00          372            NOP                     ;NOTE 1
0151 00          373            NOP
0152 00          374            NOP
0153 00          375            NOP                     ;NOTE 2
0154 00          376            NOP
```

7

```
0155 C291      377            CLR     P1.1
0157 22        378            RET
               379
               380    ;*************************************************************************
               381    ; THIS ROUTINE SENDS OUT CONTENTS OF ACCUMULATOR TO EEPROM
               382    ; without sending a START condition.
               383    ;*************************************************************************
               384
0158 7A08      385    OUT:    MOV     R2,#8           ;LOOP COUNT  -- EQUAL TO BIT COUNT
015A 33        386    OTLP:   RLC     A               ;SHIFT BIT
015B 5005      387            JNC     BITL
015D D290      388            SETB    P1.0           ;DATA = 1
015F 020164    389            JMP     OTL1            ;CONTINUE
0162 C290      390    BITL:   CLR     P1.0            ;DATA = 0
0164 D291      391    OTL1:   SETB    P1.1            ;CLOCK HI
0166 00        392            NOP                     ;NOTE 1
0167 00        393            NOP
0168 00        394            NOP
0169 00        395            NOP                     ;NOTE 2
016A 00        396            NOP
016B C291      397            CLR     P1.1            ;CLOCK LOW
016D DAEB      398            DJNZ    R2,OTLP         ;DECREMENT COUNTER
016F D290      399            SETB    P1.0            ;TURN PIN INTO INPUT
0171 00        400            NOP                     ;NOTE 1
0172 00        401            NOP                     ;NOTE 2
0173 00        402            NOP
0174 D291      403            SETB    P1.1            ;CLOCK ACK
0176 00        404            NOP                     ;NOTE 1
0177 00        405            NOP
0178 00        406            NOP
0179 00        407            NOP                     ;NOTE 2
017A 00        408            NOP
017B C291      409            CLR     P1.1
017D 22        410            RET
               411
               412    ;*************************************************************************
               413    ; THIS ROUTINE READS IN A BYTE FROM THE EEPROM
               414    ; and stores it in the accumulator
               415    ;*************************************************************************
               416
017E 7A08      417    IN:     MOV     R2,#8           ;LOOP COUNT
0180 D290      418            SETB    P1.0            ;SET DATA BIT HIGH FOR INPUT
0182 C291      419    INLP:   CLR     P1.1            ;CLOCK LOW
0184 00        420            NOP                     ;NOTE 1
0185 00        421            NOP
0186 00        422            NOP
0187 00        423            NOP
0188 00        424            NOP                     ;NOTE 2
0189 00        425            NOP
018A D291      426            SETB    P1.1            ;CLOCK HIGH
018C C3        427            CLR     C               ;CLEAR CARRY
018D 309001    428            JNB     P1.0,INL1       ;JUMP IF DATA =0
0190 B3        429            CPL     C               ;SET CARRY IF DATA =1
0191 33        430    INL1:   RLC     A                       ;ROTATE DATA INTO ACCUMULATOR
0192 DAEE      431            DJNZ    R2,INLP         ;DECREMENT COUNTER
0194 C291      432            CLR     P1.1            ;CLOCK LOW
0196 22        433            RET
               434
               435
0197 C290      436    STOP:   CLR     P1.0            ;STOP CONDITION SET DATA LOW
0199 00        437            NOP                     ;NOTE 1
019A 00        438            NOP
019B 00        439            NOP
019C 00        440            NOP                     ;NOTE 2
019D 00        441            NOP
019E D291      442            SETB    P1.1            ;SET CLOCK HI
```


© 1997 Microchip Technology Inc.

```
01A0 00        443            NOP                     ;NOTE 1
01A1 00        444            NOP
01A2 00        445            NOP
01A3 00        446            NOP                     ;NOTE 2
01A4 00        447            NOP
01A5 D290      448     SETB   P1.0            ;SET DATA HI
01A7 22        449            RET
               450
               451     ;***********************************************************************
               452     ; These routines contain special commands only for the 24LC65 SMART SERIAL
               453     ; EEPROM   *
               454     ; SET SECURE BLOCK ASSUMES
               455     ; START BLOCK 0 & BLOCK LENGTH OF 1. The
               456     ; numbers are implicit in the commands. Refer to the data sheet for details.
               457     ;*
01A8 7B80      458     SETSEC: MOV    R3,#80H         ;LOAD COMMAND AND STARTING BLOCK NUMBER
01AA 7C00      459             MOV    R4,#0
01AC 7981      460             MOV    R1,#81H         ;SET COMMAND FOR NUMBER OF BLOCKS TO SECURE
01AE 1137      461             CALL   BYTEW           ;EXECUTE
01B0 22        462             RET
               463
               464     ;*
               465     ; READ SECURE BLOCK NUMBER(S)
               466     ; RETURNS BLOCK NUMBER IN R1 AND BLOCK COUNT IN R2
               467     ; (UPPER NIBBLES WILL BE 1'S)
               468     ;*
               469
01B1 74A0      470     RDSEC:  MOV    A,#WTCMD        ;LOAD WRITE COMMMAND TO SEND ADDRESS
01B3 3120      471             CALL   OUTS            ;SEND IT
01B5 7480      472             MOV    A, #80H         ;LOAD COMMAND
01B7 3158      473             CALL   OUT             ;SEND IT
01B9 7400      474             MOV    A,#0            ;LOAD COMMAND
01BB 3158      475             CALL   OUT             ;SEND IT
01BD 74C0      476             MOV    A,#0C0H         ;LOAD COMMMAND
01BF 3158      477             CALL   OUT             ;SEND IT
01C1 317E      478             CALL   IN              ;READ STARTING BLOCK NUMBER
01C3 F9        479             MOV    R1,A            ;STORE IT
01C4 00        480             NOP                    ;NOTE 1
01C5 00        481             NOP
01C6 00        482             NOP
01C7 00        483             NOP                    ;NOTE 2
01C8 00        484             NOP
01C9 C290      485             CLR    P1.0            ;ISSUE ACK
01CB D291      486             SETB   P1.1
01CD 00        487             NOP                    ;NOTE 1
01CE 00        488             NOP
01CF 00        489             NOP
01D0 00        490             NOP                    ;NOTE 2
01D1 00        491             NOP
01D2 C291      492             CLR    P1.1
01D4 317E      493             CALL   IN              ;READ NUMBER OF BLOCKS
01D6 FA        494             MOV    R2,A            ;STORE IT
01D7 3197      495             CALL   STOP            ;SEND STOP CONDITION
01D9 22        496             RET
               497
               498
               499     ;*
               500     ; SET HIGH-ENDURANCE BLOCK NUMBER
               501     ; ASSUMES BLOCK 0
               502     ;*
01DA 7B80      503     SETHI:  MOV    R3,#80H         ;LOAD COMMAND AND BLOCK NUMBER
01DC 7C00      504             MOV    R4,#0
01DE 7900      505             MOV    R1,#0           ;SET DATA =0
01E0 1137      506             CALL   BYTEW           ;EXECUTE
01E2 22        507             RET
               508
```


© 1997 Microchip Technology Inc.

```
                509     ;*
                510     ; READ HIGH-ENDURANCE BLOCK NUMBER
                511     ; RETURNS BLOCK NUMBER IN R1 (UPPER NIBBLE WILL BE 1'S)
                512     ;*
01E3 7B80       513     READHI: MOV     R3,#80H         ;LOAD COMMAND
01E5 7C00       514     MOV     R4,#0
01E7 1201EB     515             CALL    HIEND           ;EXECUTE
01EA 22         516             RET
                517
01EB 74A0       518     HIEND:  MOV     A,#WTCMD        ;LOAD WRITE COMMMAND TO SEND ADDRESS
01ED 3120       519             CALL    OUTS            ;SEND IT
01EF EB         520             MOV     A,R3            ;GET HI BYTE ADDRESS
01F0 3158       521             CALL    OUT             ;SEND IT
01F2 EC         522             MOV     A,R4            ;GET LOW BYTE ADDRESS
01F3 3158       523             CALL    OUT             ;SEND IT
01F5 7440       524             MOV     A,#RDEND        ;LOAD READ COMMMAND
01F7 3158       525             CALL    OUT             ;SEND IT
01F9 317E       526             CALL    IN              ;READ DATA
01FB F9         527             MOV     R1,A            ;STORE DATA
01FC 3197       528             CALL    STOP            ;SEND STOP CONDITION
01FE 22         529             RET
                530
                531     ;END of 24LC65 Routines
                532     ;*************************************************************************
                533     END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```


© 1997 Microchip Technology Inc.

MICROCHIP

AN619

Interfacing the 93XX76 and 93XX86 to a PIC16C5X

Author: *Keith Pazul*
Microchip Technology Inc.

INTRODUCTION

Microchip's newest offerings in the 3-wire Serial EEPROM family are the 93XX76 and 93XX86 devices. These are 8 Kbit and 16 Kbit versions of our 93 series Serial EEPROMs with a couple of differences from the standard 93XX46, 93XX56 and 93XX66 devices:

1. These devices have a Program Enable (PE) pin, which is used as a hardware write protect. Tying the PE pin to Vcc enables the array to be written to, while tying it to Vss prohibits the array from being written to. There is an internal pull-up resistor on the PE pin that allows the array to be written to when there is no connection to the PE pin.
2. The internal write cycle begins on the rising edge of the last data bit (D0) for the 93XX76 and 93XX86 devices, where the existing 93XX46, 93XX56 and 93XX66 devices begin their internal write cycle on the falling edge of Chip Select ($\overline{CS}$) after the last data bit has been sent. Software written for the 93XX46, 93XX56 and 93XX66 devices will still be compatible, however the internal write cycle will begin slightly earlier on the new 93XX76 and 93XX86 devices. The only changes to the code required are the number of address bits necessary to access the larger memory arrays.

The code that is written for this application note was taken from AN560 and the only modifications were the number of address bits output. Please refer to AN560 for more details about the original code. If the reader is new to Serial EEPROMs, please refer to application note AN536 for a more basic tutorial. This application note is intended to provide the engineer with stand-alone code modules for accomplishing all of the necessary functions required to utilize these devices in an application. See Figure 1 for a schematic of how the PIC16C54 was connected to the 93XX76 and 93XX86.

Although these devices are called 3-wire devices, they have essentially 4 I/O pins:

CS	Chip Select
CLK	Clock
DI	Data In
DO	Data Out

These devices use a series of commands to accomplish standard memory functions. These are READ, WRITE, EWEN, ERASE, ERAL, WRAL, EWDS. These functions are discussed in more detail in the datasheets for any of the 93 series devices.

The following programs are included in this application note and are fully functional stand-alone modules. The source code for these modules can be downloaded from the Microchip BBS. They are located in the MEM_APPS file library as one zipped up file, AN619.zip. Please consult page 8-1 of the *1995/1996 Non-Volatile Memory Products Data Book* or page 6-3 of the *1995/1996 Embedded Control Handbook Update* for more information on how to use the Microchip BBS.

The following are the programs included in this application note, along with the subroutines they call:

Byte Read Program (7686byrd.asm)

- Start Bit Routine
- Bin In Routine
- Receive Data routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine

Byte Write Program (7686bywr.asm)

- Delay Routine
- Start Bit Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Erase/Write Enable Routine (EWEN)
- Byte Write Routine
- Erase/Write Disable Routine (EWDS)

7

© 1997 Microchip Technology Inc.

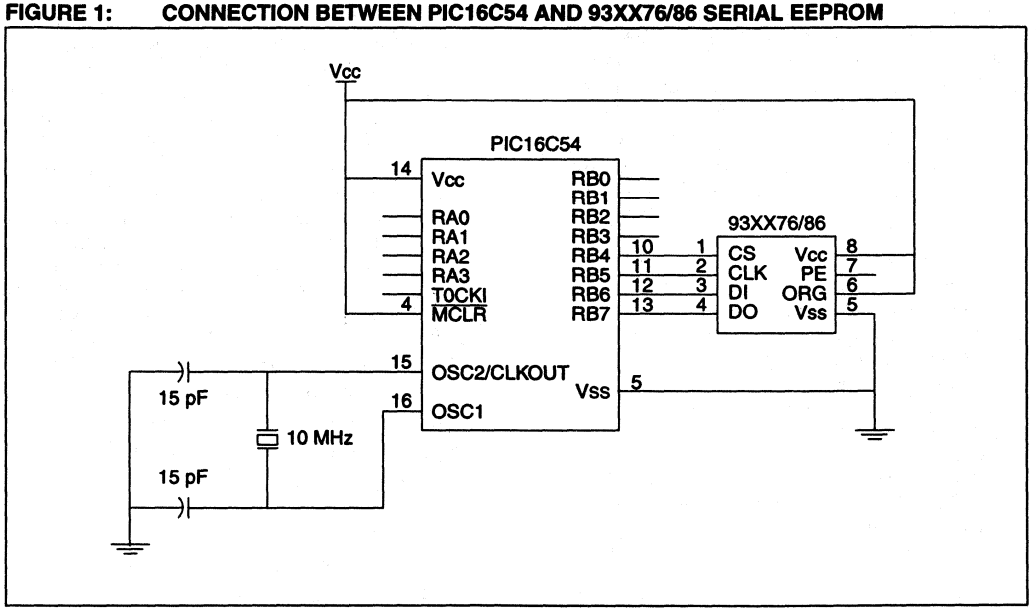
Byte Write with Data Polling Program (7686dpol.asm)

- Data Polling Delay Routine
- Start Bit Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Erase/Write Enable Routine (EWEN)
- Write Routine
- Erase/WRite Disable Routine (EWDS)

Sequential Read Program (7686seqr.asm)

- Delay Routine
- Start Bit Routine
- Bit In Routine
- Receive Data Routine
- Bit Out Routine
- Transmit Data Routine
- Power-up Routine
- Read Routine

FIGURE 1: CONNECTION BETWEEN PIC16C54 AND 93XX76/86 SERIAL EEPROM



© 1997 Microchip Technology Inc.

Interfacing Motorola 68HC11 to Microchip SPI™ Serial EEPROMS

Author: *Shannon Poulin*
Microchip Technology Inc.

INTRODUCTION

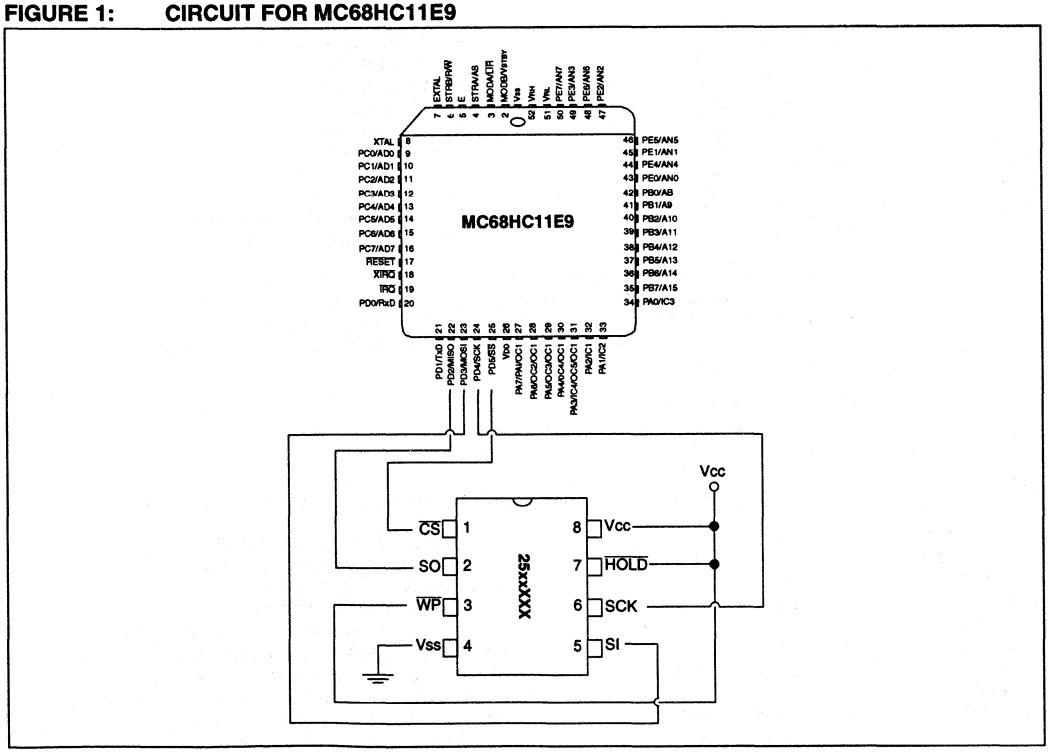
There are many different microcontrollers on the market today that are being used in embedded control applications. Many of these embedded control systems need non-volatile memory. Because of their small footprint, byte level flexibility, low I/O pin requirement, low power consumption, and low cost, Serial EEPROMs are a popular choice for non-volatile storage.

Microchip addresses these needs by offering a full line of Serial EEPROMs covering industry standard serial communication protocols for 2-wire, 3-wire, and SPI communication. Serial EEPROM devices are available in a variety of densities, operational voltage ranges, and packaging options.

Microchip realizes that its customer base is very broad, and because of this, different microcontrollers are used to interface to SPI Serial EEPROMs. One of the microcontrollers used in these applications is the Motorola 68HC11. In order to simplify the design process, Microchip has written an assembly code routine to communicate with our SPI parts that is verified and tested to function properly on a 68HC11.

Figure 1 describes the hardware schematic for the interface between Microchip's SPI devices and the Motorola 68HC11E9. The schematic shows the connections necessary between the microcontroller and the serial EEPROM, and the software was written assuming these connections. Appendix A contains a listing of the SPI source code.

FIGURE 1: CIRCUIT FOR MC68HC11E9



SPI is a trademark of Motorola.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: SOURCE CODE

```
an646.asm               Assembled with CASM   03/19/1997  16:10  PAGE 1


                         1  ;*****************************************************************
                         2  ;* This code demonstrates how the Microchip SPI Devices
                         3  ;* can be interfaced to the SPI port of the 68HC11 microcontroller.
                         4  ;* The interface uses the three SPI lines (SCK,MOSI and MISO) on
                         5  ;* the 68HC11 Microcontroller for the clock (SCK), data out (SO)
                         6  ;* and data in (SI). A chip select (CS) is generated with a
                         7  ;* general purpose port line (PD5). The 68HC11 is placed into the
                         8  ;* master mode which allows use of the slave select input (SS) for
                         9  ;* other things.  The code uses the 0 mode (CPOL = 0,  CPHA = 0)
                        10  ;* of operation to define clock polarity and phase. MODE 3
                        11  ;* (CPOL = 1, CPHA = 1) could also have been used.  The  baud rate
                        12  ;* set at one eighth of the clock frequency (SPR0 = 0,  SPR1 = 0)
                        13  ;*
                        14  ;* This code shows a read and write operation, as well  as data
                        15  ;* polling, setting the write enable latch, clearing the  write
                        16  ;* enable latch, writing and reading the block protect
                        17  ;* register
                        18  ;*****************************************************************
                        19
                        20  ; LOAD VALUES FOR USE LATER
                        21
                        22
0000                    23  MASKCS  EQU     $20             ;MASK THE CHIP ENABLE BIT
0000                    24  DDRD    EQU     $09             ;PORTD DATA DIRECTION REGISTER
0000                    25  PORTD   EQU     $08             ;PORT D ADDRESS
0000                    26  SPCR    EQU     $28             ;SPI CONTROL REGISTER
0000                    27  SPSR    EQU     $29             ;SPI STATUS REGISTER
0000                    28  SPDR    EQU     $2A             ;SPI DATA REGISTER
0000                    29  ADDRLO  EQU     $80             ;LOW ADDRESS LOCATION
0000                    30  ADDRHI  EQU     $81             ;HIGH ADDRESS LOCATION
0000                    31  RDATA   EQU     $82             ;READ DATA SCRATCH PAD LOCATION
0000                    32  SDATA   EQU     $83             ;SEND DATA SCRATCH PAD LOCATION
                        33
                        34  ;*****************************************************
                        35  ;* SET RESET VECTOR TO THE BEGINNING OF THE PROGRAM
                        36  ;*****************************************************
                        37  ;         ORG     $FFFE           ;RESET VECTOR TO PROGRAM ENTRY POINT
                        38  ;         FDB     $E000
                        39
                        40  ;*********************************
                        41  ;* BEGINNING OF PROGRAM EXECUTION
                        42  ;*********************************
                        43
B600                    44    ORG     $B600           ;BEGINNING ADDRESS OF EXECUTABL E CODE
                        45
B600 8E00FF             46  BEGIN   LDS     #$00FF          ;INITIALIZE STACK POINTER
B603 CE1000             47    LDX     #$1000          ;INITIALIZE PAGE OFFSET LOCATION
B606 863F               48    LDAA    #$3F            ;LOAD ACCUMULATOR WITH 3F HEX
B608 A709               49    STAA    DDRD,X          ;MAKE ALL PORTD PINS OUTPUTS
B60A 8650               50    LDAA    #$50            ;SPI (MODE 0,0), USE #$5C FOR (MODE 1,1)
B60C A728               51    STAA    SPCR,X          ;STORE VALUE IN THE SPI CONTROL REG
B60E A629               52    LDAA    SPSR,X          ;READ MICRO SPI STATUS REG, CLEAR SPIF
B610 8655               53    LDAA    #$55            ;LOAD ACCUMULATOR A WITH 2E HEX
B612 9780               54    STAA    ADDRLO          ;STORE THIS VALUE IN ADDRESS LOW
B614 8600               55    LDAA    #$00            ;LOAD ACCUMULATOR A WITH 00 HEX
```


© 1997 Microchip Technology Inc.

```
B616 9781        56    STAA    ADDRHI          ;STORE THIS VALUE IN ADDRESS HIGH
                 57  ; SEND THE WRITE ENABLE SEQUENCE (WREN)
B618 BDB6EE      58    JSR     CSLOW           ;JUMP TO THE CHIP SELECT LOW  ROUTINE
B61B 8606        59    LDAA    #$06            ;LOAD THE WRITE SELECT COMMAND
B61D 9783        60    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B61F BDB6E1      61    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
B622 BDB6F2      62    JSR     CSHIGH          ;JUMP TO THE CHIP SELECT HIGH ROUTINE
                 63  ; SEND THE WRITE STATUS REGISTER SEQUENCE (WRSR)
B625 BDB6EE      64    JSR     CSLOW           ;JUMP TO THE CHIP SELECT LOW ROUTINE
B628 8601        65    LDAA    #$01            ;LOAD THE WRITE STATUS  REGISTER COMMAND
B62A 9783        66    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B62C BDB6E1      67    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
B62F 8600        68    LDAA    #$00            ;LOAD ZERO'S INTO THE STATUS REGISTER
B631 9783        69    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B633 BDB6E1      70    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
B636 BDB6F2      71    JSR     CSHIGH          ;JUMP TO THE CHIP SELECT HIGH ROUTINE
                 72  ; WAIT THE REQUIRED 5mS FOR THE WRITE CYCLE TIMER Twc
B639 BDB6F6      73    JSR     DELAY           ;JUMP TO THE DELAY SUBROUTINE
                 74  ; SEND THE WRITE ENABLE SEQUENCE (WREN)
B63C BDB6EE      75    JSR     CSLOW           ;JUMP TO THE CHIP SELECT LOW SUBROUTINE
B63F 8606        76    LDAA    #$06            ;LOAD THE WRITE ENABLE COMMAND
B641 9783        77    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B643 BDB6E1      78    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
B646 BDB6F2      79    JSR     CSHIGH          ;JUMP TO THE CHIP SELECT HIGH ROUTINE
                 80  ; SEND THE READ STATUS REGISTER SEQUENCE (RDSR)
B649 BDB6EE      81    JSR     CSLOW           ;JUMP TO THE CHIP SELECT LOW SUBROUTINE
B64C 8605        82    LDAA    #$05            ;LOAD THE READ STATUS REGISTER COMMAND
B64E 9783        83    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B650 BDB6E1      84    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
B653 BDB6E1      85    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE (READ)
B656 BDB6F2      86    JSR     CSHIGH          ;JUMP TO THE CHIP SELECT HIGH ROUTINE
                 87  ; SEND THE WRITE SEQUENCE (WRITE)
B659 BDB6EE      88    JSR     CSLOW           ;JUMP TO THE CHIP SELECT LOW SUBROUTINE
B65C 8602        89    LDAA    #$02            ;LOAD THE WRITE COMMAND
B65E 9783        90    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B660 BDB6E1      91    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
                 92  ; ***COMMENT OUT FOR USE WITH 25XX010, 25XX020, OR 25XX040*************
B663 9681        93    LDAA    ADDRHI          ;LOAD ACCA WITH ADDRHI
B665 9783        94    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B667 BDB6E1      95    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
                 96  ; ********************************************************************
B66A 9680        97    LDAA    ADDRLO          ;LOAD ACCA WITH ADDRLO
B66C 9783        98    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B66E BDB6E1      99    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
B671 86AA       100    LDAA    #$AA            ;LOAD ACCA WITH HEX AA
B673 9783       101    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B675 BDB6E1     102    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
B678 BDB6F2     103    JSR     CSHIGH          ;JUMP TO THE CHIP SELECT HIGH ROUTINE
                104  ; PERFORM DATA POLLING (RDSR BIT 0)
B67B BDB6EE     105    JSR     CSLOW           ;JUMP TO THE CHIP SELECT LOW ROUTINE
B67E 8605       106    LDAA    #$05            ;LOAD THE READ STATUS REGISTER COMMAND
B680 9783       107    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B682 BDB6E1     108    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
                109  ;
B685 8630       110    LDAA    #$30            ;GIVE THE SPI DEVICE TIME TO SET WIP
B687 4A         111    LOOP    DECA            ;DECREMENT ACCUMULATOR A
B688 26FD       112    BNE     LOOP            ;BRANCH IF NOT EQUAL TO LOOP
                113  ;
B68A BDB6E1     114  POLLING JSR     OUTBYTE   ;READ THE DATA IN THE STATUS REGISTER
B68D 8601       115    LDAA    #$01            ;LOAD ACCA WITH 1 HEX
B68F 9482       116    ANDA    RDATA           ;AND ACCA WITH THE RECEIVED DATA
B691 26F7       117    BNE     POLLING         ;BRANCH IF NOT = TO POLLING
B693 BDB6F2     118    JSR     CSHIGH          ;JUMP TO THE CHIP SELECT HIGH ROUTINE
                119  ; SEND THE READ SEQUENCE, READ ADDRESS 0X55
B696 BDB6EE     120    JSR     CSLOW           ;JUMP TO THE CHIP SELECT LOW ROUTINE
B699 8603       121    LDAA    #$03            ;LOAD THE READ COMMAND
```


© 1997 Microchip Technology Inc.

```
B69B 9783       122    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B69D BDB6E1     123    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
                124  ; ***COMMENT OUT FOR USE WITH 25XX010, 25XX020, OR 25XX040*************
B6A0 9681       125    LDAA    ADDRHI          ;LOAD ACCA WITH ADDRHI
B6A2 9783       126    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B6A4 BDB6E1     127    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
                128  ; ***************************************************************
B6A7 9680       129    LDAA    ADDRLO          ;LOAD ACCA WITH ADDRLO
B6A9 9783       130    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B6AB BDB6E1     131    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
B6AE BDB6E1     132    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE (READ)
B6B1 BDB6F2     133    JSR     CSHIGH          ;JUMP TO THE CHIP SELECT HIGH ROUTINE
                134  ; SEND THE WRITE ENABLE SEQUENCE (WREN)
B6B4 BDB6EE     135    JSR     CSLOW           ;JUMP TO THE CHIP SELECT LOW ROUTINE
B6B7 8606       136    LDAA    #$06            ;LOAD THE WRITE ENABLE COMMAND
B6B9 9783       137    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B6BB BDB6E1     138    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
B6BE BDB6F2     139    JSR     CSHIGH          ;JUMP TO THE CHIP SELECT HIGH ROUTINE
                140  ; SEND THE WRITE DISABLE SEQUENCE (WRDI)
B6C1 BDB6EE     141    JSR     CSLOW           ;JUMP TO THE CHIP SELECT LOW ROUTINE
B6C4 8604       142    LDAA    #$04            ;LOAD THE WRITE DISABLE COMMAND
B6C6 9783       143    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B6C8 BDB6E1     144    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
B6CB BDB6F2     145    JSR     CSHIGH          ;JUMP TO THE CHIP SELECT HIGH ROUTINE
                146  ; SEND THE READ STATUS REGISTER SEQUENCE
B6CE BDB6EE     147    JSR     CSLOW           ;JUMP TO THE CHIP SELECT LOW ROUTINE
B6D1 8605       148    LDAA    #$05            ;LOAD THE READ STATUS REGISTER COMMAND
B6D3 9783       149    STAA    SDATA           ;STORE THE VALUE IN THE SDATA LOCATION
B6D5 BDB6E1     150    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE
B6D8 BDB6E1     151    JSR     OUTBYTE         ;JUMP TO THE SEND BYTE ROUTINE (READ)
B6DB BDB6F2     152    JSR     CSHIGH          ;JUMP TO THE CHIP SELECT HIGH ROUTINE
B6DE 7EB600     153    JMP     BEGIN           ;LOOP UNTIL RESET
                154
                155
                156  ;*************************************************************************
                157  ;* THIS ROUTINE SENDS A BYTE OUT TO THE SPI PART. THEN IT READS WHAT IS SENT
                158  ;* BACK ON THE DO PIN.  DATA IS SENT OUT FROM THE 68HC11 ON THE MOSI PIN AND
                159  ;*  READ IN ON THE MISO PIN.  THE CLOCK IS GENERATED BY THE SCK PIN. THIS
                160  ;*  POLLS THE SPIF BIT IN THE SPSR TO SEE IF THE TRANSFER HAS BEEN COMPLETED
                161  ;*  WHEN COMPLETE THE DATA READ IS AVAILABLE IN THE SPDR SPI DATA REGISTER
                162  ;*************************************************************************
                163
B6E1 9683       164  OUTBYTE LDAA    SDATA              ;LOAD A WITH WHAT'S IN SDATA
B6E3 A72A       165    STAA    SPDR,X                   ;SEND BYTE OUT PART
B6E5 1F2980FC   166  WAIT1   BRCLR   SPSR,X,$80,WAIT1   ;WAIT FOR BYTE TO BE SENT
B6E9 A62A       167    LDAA    SPDR,X                   ;READ BYTE BACK INTO ACCA
B6EB 9782       168    STAA    RDATA                    ;STORE THE READ IN LOCATION RDATA
B6ED 39         169    RTS                              ;RETURN FROM SUBROUTINE
                170
                171  ;*********************
                172  ;* SET CE (PD5) LOW
                173  ;*********************
                174
B6EE 1D0820     175  CSLOW   BCLR    PORTD,X,MASKCS          ;SET CE LOW
B6F1 39         176    RTS                              ;RETURN FROM SUBROUTINE
                177
                178  ;*********************
                179  ;* SET CE (PD5) HIGH
                180  ;*********************
                181
B6F2 1C0820     182  CSHIGH  BSET    PORTD,X,MASKCS          ;SET CE HIGH
B6F5 39         183    RTS                              ;RETURN FROM SUBROUTINE
                184
                185  ;*********************
                186  ;* DELAY SUBROUTINE
                187  ;*********************
```


© 1997 Microchip Technology Inc.

```
                188  ;DELAY 250 (0xFA) * 400Ns X 50 (0x32)
                189
B6F6 8632       190  DELAY   LDAA    #$32                ;LOAD ACCA WITH HEX 32
B6F8 C6FA       191  DEC2    LDAB    #$FA                ;LOAD ACCB WITH FA HEX
B6FA 5A         192  DEC1    DECB                        ;DECREMENT ACCB
B6FB 26FD       193    BNE     DEC1                   ;BRANCH IF NOT EQUAL TO DEC1
B6FD 4A         194    DECA                           ;DECREMENT ACCA
B6FE 26F8       195    BNE     DEC2                   ;BRANCH IF NOT EQUAL TO DEC2
B700 39         196    RTS                            ;RETURN FROM SUBROUTINE
                197
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Interfacing Microchip PIC16C64/74 to Microchip SPI™ Serial EEPROMs

Author: *Shannon Poulin*
Microchip Technology Inc.

INTRODUCTION

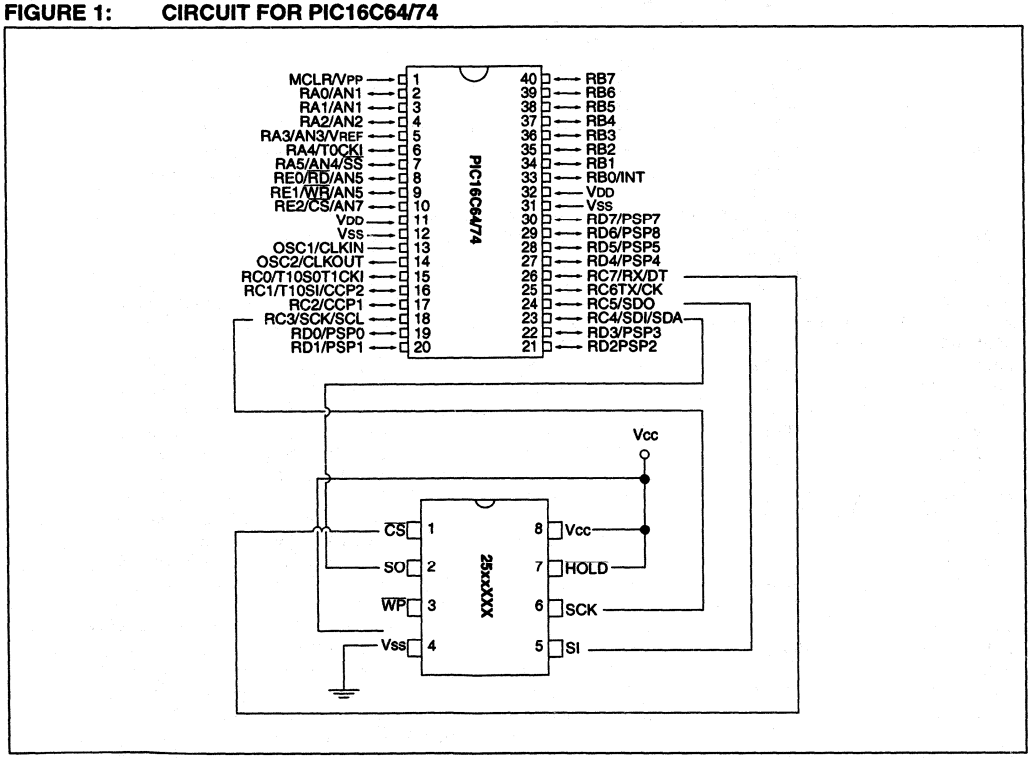
There are many different microcontrollers on the market today that are being used in embedded control applications. Many of these embedded control systems need non-volatile memory. Because of their small footprint, byte level flexibility, low I/O pin requirement, low power consumption, and low cost, serial EEPROMs are a popular choice for non-volatile storage.

Microchip addresses these needs by offering a full line of serial EEPROMs covering industry standard serial communication protocols for two-wire, three-wire, and SPI communication. Serial EEPROM devices are available in a variety of densities, operational voltage ranges, and packaging options.

This application note provides assistance and source code to ease the design process of interfacing a Microchip mid-range PIC16C6X/7X microcontroller and a Microchip SPI serial EEPROM. The hardware SPI port on the microcontroller provides a simple three-wire connection to the EEPROM and no external "glue" logic is required

Figure 1 describes the hardware schematic for the interface between Microchip's SPI devices and the Microchip PIC16C64/74 Microcontroller. The schematic shows the connections necessary between the microcontroller and the serial EEPROM, and the software was written assuming these connections. Appendix A contains a listing of the SPI source code.

FIGURE 1: CIRCUIT FOR PIC16C64/74



SPI is a trademark of Motorola.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A: SOURCE CODE

```
MPASM 01.40.01 Intermediate     AN647.ASM   3-20-1997  9:13:03         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST P=16C74
                     00002 ; Port C pin descriptions
                     00003 ; SCK bit = 3
                     00004 ; SDI bit = 4
                     00005 ; SDO bit = 5
                     00006 ; CS bit  = 7
                     00007 ;
                     00008 ; 10MHz crystal is being used, thus each instruction cycle = 400nS
                     00009 ;
                     00010 ;******************RAM register definitions*********************
  00000020           00011 rxdata  equ     20h
  00000021           00012 txdata  equ     21h
  00000022           00013 addr    equ     22h
  00000023           00014 loops   equ     23h
  00000024           00015 outbyte equ     24h
  00000025           00016 temp1   equ     25h
  00000026           00017 temp2   equ     26h
                     00018 ;******************Bit definitions*******************************
  00000007           00019 CS      equ     7
                     00020 ;******************Include file*********************************
                     00021         include "p16c74.inc"  ; this is the include file for a PIC16C74
                     00001         LIST
                     00002 ; P16C74.INC Standard Header File, Ver. 1.00 Microchip Technology, Inc.
                     00318         LIST
                     00022 ;*************************************************************
0000                 00023         org     0x000           ; set the reset vector
0000 2801            00024         goto    start           ; go to the beginning of main
                     00025
                     00026 ;!!!!!!!!!!!!!!!!!!!!Begin Main Program!!!!!!!!!!!!!!!!!!!!!!!!!
0001 1283            00027 start   bcf     STATUS,RP0      ; set to bank 0
0002 0187            00028         clrf    PORTC           ; initialize portc to 0
0003 1683            00029         bsf     STATUS,RP0      ; set to bank 1
0004 3010            00030         movlw   0x10            ; all bits are outputs except SDI
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0005 0087            00031         movwf   TRISC           ; move the value to TRIS portc
0006 1283            00032         bcf     STATUS,RP0      ; set to bank0
0007 1787            00033         bsf     PORTC,CS        ; make the chip select is high
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0008 018C            00034         clrf    PIE1            ; disable peripheral interrupts
0009 018B            00035         clrf    INTCON          ; disables all interrupts
000A 1283            00036         bcf     STATUS,RP0      ; set to bank 0
000B 0194            00037         clrf    SSPCON          ; clear SSP control register
000C 3031            00038         movlw   0x31            ; set up spi port, SPI master,
000D 0094            00039         movwf   SSPCON          ; clk/16, ckp=1 (mode 1,1)
000E 3055            00040         movlw   0x55            ; put starting address 55 in
000F 00A2            00041         movwf   addr            ; addr for later use
                     00042 ;Send the write enable sequence (WREN)
0010 1387            00043         bcf     PORTC,CS        ; clear the chip select (active)
0011 3006            00044         movlw   0x06            ; load WREN sequence
0012 00A4            00045         movwf   outbyte         ; store in RAM location outbyte
0013 205F            00046         call    output          ; call the SPI output routine
0014 1787            00047         bsf     PORTC,CS        ; set the chip select line
                     00048 ;Send the write status register sequence (WRSR)
0015 1387            00049         bcf     PORTC,CS        ; clear the chip select line
```


© 1997 Microchip Technology Inc.

```
0016 3001           00050         movlw   0x01            ; clear all status register
0017 00A4           00051         movwf   outbyte         ; store in RAM location outbyte
0018 205F           00052         call    output          ; call the SPI output routine
0019 3000           00053         movlw   0x00            ; load up zero to send
001A 00A4           00054         movwf   outbyte         ; store in RAM location outbyte
001B 205F           00055         call    output          ; call the SPI output routine
001C 1787           00056         bsf     PORTC,CS        ; set the chip select line
                    00057 ;Wait the required 5mS for the write cycle timer Twc
001D 2068           00058         call    delay           ; call the delay subroutine
                    00059 ;Send the write enable sequence (WREN)
001E 1387           00060         bcf     PORTC,CS        ; clear the chip select (active)
001F 3006           00061         movlw   0x06            ; load WREN sequence
0020 00A4           00062         movwf   outbyte         ; store in RAM location outbyte
0021 205F           00063         call    output          ; call the SPI output routine
0022 1787           00064         bsf     PORTC,CS        ; set the chip select line
                    00065 ;Send the read status register sequence (RDSR)
0023 1387           00066         bcf     PORTC,CS        ; clear the chip select line
0024 3005           00067         movlw   0x05            ; load RDSR sequence
0025 00A4           00068         movwf   outbyte         ; store in RAM location outbyte
0026 205F           00069         call    output          ; call the SPI output routine
0027 205F           00070         call    output          ; read the data in status reg.
0028 1787           00071         bsf     PORTC,CS        ; set the chip select line
                    00072 ;Send the write sequence (WRITE)
0029 1387           00073         bcf     PORTC,CS        ; clear the chip select line
002A 3002           00074         movlw   0x02            ; load WRITE sequence
002B 00A4           00075         movwf   outbyte         ; store in RAM location outbyte
002C 205F           00076         call    output          ; call the SPI output routine
                    00077 ;****Comment out for use with 25xx010, 25xx020 or 25xx040********
002D 3000           00078         movlw   0x00            ; load high address byte
002E 00A4           00079         movwf   outbyte         ; store in RAM location outbyte
002F 205F           00080         call    output          ; call the SPI output routine
                    00081 ;*****************************************************************
0030 0822           00082         movf    addr,W          ; move the address into w
0031 00A4           00083         movwf   outbyte         ; store in RAM location outbyte
0032 205F           00084         call    output          ; call the SPI output routine
0033 30AA           00085         movlw   0xAA            ; load data AA into w
0034 00A4           00086         movwf   outbyte         ; store in RAM location outbyte
0035 205F           00087         call    output          ; call the SPI output routine
0036 1787           00088         bsf     PORTC,CS        ; set the chip select line
                    00089 ;Perform data polling (RDSR bit 0)
0037 1387           00090         bcf     PORTC,CS        ; clear the chip select line
0038 3005           00091         movlw   0x05            ; load RDSR sequence
0039 00A4           00092         movwf   outbyte         ; store in RAM location outbyte
003A 205F           00093         call    output          ; call the SPI output routine
                    00094 ;
003B 3030           00095         movlw   0x30            ; give the spi device time to
003C 00A3           00096         movwf   loops           ; set the WIP bit in the
Message[305]: Using default destination of 1 (file).
003D 0BA3           00097 wait    decfsz  loops           ; status register before coming
003E 283D           00098         goto    wait            ; back and doing data polling
                    00099 ;
003F 205F           00100 polling call    output          ; read the data in status reg.
0040 1820           00101         btfsc   rxdata,0        ; test the WIP bit in status reg.
0041 283F           00102         goto    polling         ; WIP clear, loop until WIP set
0042 1787           00103         bsf     PORTC,CS        ; set the chip select line
                    00104 ;Send read sequence (READ), read address 0x55
0043 1387           00105         bcf     PORTC,CS        ; clear the chip select line
0044 3003           00106         movlw   0x03            ; load READ sequence
0045 00A4           00107         movwf   outbyte         ; store in RAM location outbyte
0046 205F           00108         call    output          ; call the SPI output routine
                    00109 ;****Comment out for use with 25xx010, 25xx020 or 25xx040********
0047 3000           00110         movlw   0x00            ; load high address byte
0048 00A4           00111         movwf   outbyte         ; store in RAM location outbyte
0049 205F           00112         call    output          ; call the SPI output routine
                    00113 ;*****************************************************************
004A 0822           00114         movf    addr,W          ; move the address into w
```


© 1997 Microchip Technology Inc.

```
004B 00A4           00115         movwf   outbyte         ; store in RAM location outbyte
004C 205F           00116         call    output          ; call the SPI output routine
004D 205F           00117         call    output          ; call output to read 1 byte
004E 1787           00118         bsf     PORTC,CS        ; set the chip select line
                    00119 ;Send the write enable sequence (WREN)
004F 1387           00120         bcf     PORTC,CS        ; clear the chip select (active)
0050 3006           00121         movlw   0x06            ; load WREN sequence
0051 00A4           00122         movwf   outbyte         ; store in RAM location outbyte
0052 205F           00123         call    output          ; call the SPI output routine
0053 1787           00124         bsf     PORTC,CS        ; set the chip select line
                    00125 ;Send the write disable sequence (WRDI)
0054 1387           00126         bcf     PORTC,CS        ; clear the chip select (active)
0055 3004           00127         movlw   0x04            ; load WRDI sequence
0056 00A4           00128         movwf   outbyte         ; store in RAM location outbyte
0057 205F           00129         call    output          ; call the SPI output routine
0058 1787           00130         bsf     PORTC,CS        ; set the chip select line
                    00131 ;Send the read status register sequence (RDSR)
0059 1387           00132         bcf     PORTC,CS        ; clear the chip select line
005A 3005           00133         movlw   0x05            ; load RDSR sequence
005B 00A4           00134         movwf   outbyte         ; store in RAM location outbyte
005C 205F           00135         call    output          ; call the SPI output routine
005D 205F           00136         call    output          ; read the data in status reg.
                    00137 ;*******Go back to main routine*********************************
005E 2801           00138         goto    start
                    00139
                    00140 ;******************SPI output subroutine************************
005F 0824           00141 output  movf    outbyte,W       ; move outbyte into w
0060 0093           00142         movwf   SSPBUF          ; place data in send buffer
0061 1683           00143 loop1   bsf     STATUS,RP0      ; set to bank 1
Message[302]: Register in operand not in bank 0.  Ensure that bank bits are correct.
0062 1C14           00144         btfss   SSPSTAT,BF      ; has data been received?
0063 2861           00145         goto    loop1           ; loop if not received yet
0064 1283           00146         bcf     STATUS,RP0      ; set to bank 0
0065 0813           00147         movf    SSPBUF,W        ; empty the receive buffer
0066 00A0           00148         movwf   rxdata          ; put received byte into rxdata
0067 3400           00149         retlw   0               ; return from subroutine
                    00150
                    00151 ; 250 x 400nS x 50 = 5mS (plus overhead)
0068 3032           00152 delay   movlw   0x32            ; move 50 decimal into w
0069 00A5           00153         movwf   temp1           ; move 50 decimal into temp1
006A 30FA           00154 dec1    movlw   0xFA            ; move 250 decimal into w
006B 00A6           00155         movwf   temp2           ; move 250 decimal into temp2
006C 0BA6           00156 dec2    decfsz  temp2,1         ; decrement temp2, skip if zero
006D 286C           00157         goto    dec2            ; goto decrement 2 if not zero
006E 0BA5           00158         decfsz  temp1,1         ; decrement temp1, skip if zero
006F 286A           00159         goto    dec1            ; goto decrement 1 if not zero
0070 3400           00160         retlw   0               ; return both locations = 0
                    00161
                    00162
                    00163         END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX X---------------

All other memory blocks unused.

Program Memory Words Used:   113
Program Memory Words Free:  3983


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     4 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

Interfacing Microchip PIC16C54 to Microchip SPI™ Serial EEPROMs

Author: *Shannon Poulin*
Microchip Technology Inc.

INTRODUCTION

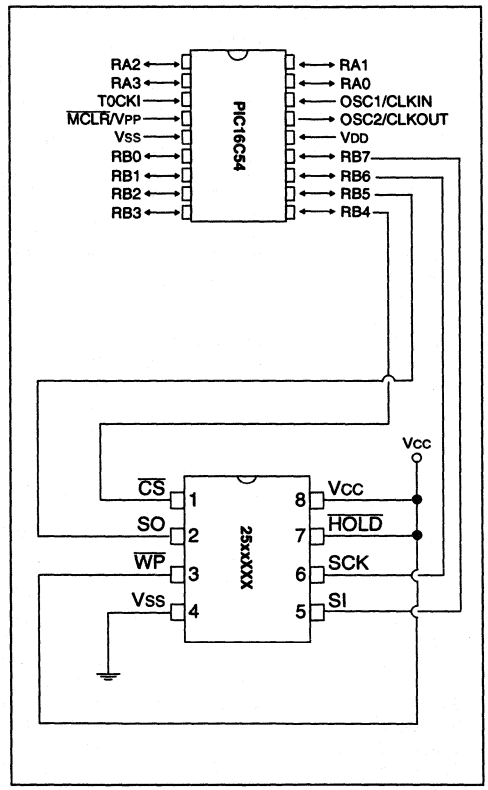
There are many different microcontrollers on the market today that are being used in embedded control applications. Many of these embedded control systems need non-volatile memory. Because of their small footprint, byte level flexibility, low I/O pin requirement, low power consumption, and low cost, serial EEPROMs are a popular choice for non-volatile storage.

Microchip addresses these needs by offering a full line of serial EEPROMs covering industry standard serial communication protocols for 2-wire, 3-wire, and SPI communication. Serial EEPROM devices are available in a variety of densities, operational voltage ranges and packaging options.

This application note provides assistance and source code to ease the design process of interfacing a Microchip PIC16C54 microcontroller and Microchip SPI serial EEPROMs. The connections will be made using port pins from the microcontroller to interface to the memory device. Bit manipulation will be used to provide the proper logic necessary to write and read the part.

Figure 1 describes the hardware schematic for the interface between Microchip's SPI devices and the Microchip PIC16C54 Microcontroller. The schematic shows the connections necessary between the microcontroller and the serial EEPROM, and the software was written assuming these connections. Appendix A contains a listing of the SPI source code.

FIGURE 1: CIRCUIT FOR PIC16C54



7

SPI is a trademark of Motorola.

© 1997 Microchip Technology Inc.

Please check the Microchip BBS for the latest version of the source code. Microchip's Worldwide Web Address: www.microchip.com; Bulletin Board Support: MCHIPBBS using CompuServe® (CompuServe membership not required).

APPENDIX A:

```
MPASM 01.40.01 Intermediate     AN648.ASM   3-20-1997  15:04:06         PAGE  1


LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001         LIST P=16C54
                     00002 ; Port B pin descriptions
                     00003 ; PIC16C54                        SPI Device
                     00004 ; Pin #         Description       Pin #       Description
                     00005 ;  13           PORTB,7             5         SI, serial data in
                     00006 ;  12           PORTB,6             6         SCK, serial clock
                     00007 ;  11           PORTB,5             2         SO, serial data out
                     00008 ;  10           PORTB,4             1         not CS, not chip select
                     00009 ;
                     00010 ;******************RAM register definitions*********************
  0000000B           00011 count   equ     0x0B              ; count variable location
  0000000C           00012 outbyte equ     0x0C              ; outbyte storage variable
  0000000D           00013 temp1   equ     0x0D              ; temp location 1
  0000000E           00014 temp2   equ     0x0E              ; temp location 2
  0000000F           00015 rxdata  equ     0x0F              ; location for received data
                     00016 ;******************Bit definitions*****************************
  00000007           00017 si      equ     7                 ; SPI data in = portb pin 7
  00000006           00018 sck     equ     6                 ; SPI clock = portb pin 6
  00000005           00019 so      equ     5                 ; SPI data out = portb pin 5
  00000004           00020 cs      equ     4                 ; SPI chip select = portb pin 4
                     00021 ;******************Include file*******************************
                     00022         include "p16c5x.inc"      ; include file for a PIC16C54
                     00001         LIST
                     00002 ; P16C5X.INC  Standard Header File, Version 3.30 Microchip Technology
                     00224         LIST
                     00023 ;*****************************************************************
01FF                 00024         org     01ffh             ; set the reset vector
01FF 0A01            00025         goto    start             ; go to the beginning of main
0000                 00026         org     000h              ;
0000 0A01            00027         goto    start             ;
                     00028 ;!!!!!!!!!!!!!!!!!!!!!!Begin Main Program!!!!!!!!!!!!!!!!!!!!!!!!!
0001 0C20            00029 start   movlw   0x20              ; move 20 hex into w
0002 0006            00030         tris    PORTB             ; portb all outputs except pin 5
0003 0586            00031         bsf     PORTB,cs          ; set the chip select line
0004 04C6            00032         bcf     PORTB,sck         ; clear the clock line
0005 04E6            00033         bcf     PORTB,si          ; clear the data in line
                     00034 ;Send the write enable sequence (WREN)
0006 0486            00035         bcf     PORTB,cs          ; clear the chip select (active)
0007 0C06            00036         movlw   0x06              ; load WREN sequence
0008 002C            00037         movwf   outbyte           ; store in RAM location outbyte
0009 0955            00038         call    output            ; call the SPI output routine
000A 0586            00039         bsf     PORTB,cs          ; set the chip select line
                     00040 ;Send the write status register sequence (WRSR)
000B 0486            00041         bcf     PORTB,cs          ; clear the chip select line
000C 0C01            00042         movlw   0x01              ; clear all status register
000D 002C            00043         movwf   outbyte           ; store in RAM location outbyte
000E 0955            00044         call    output            ; call the SPI output routine
000F 0C00            00045         movlw   0x00              ; load up zero to send
0010 002C            00046         movwf   outbyte           ; store in RAM location outbyte
0011 0955            00047         call    output            ; call the SPI output routine
0012 0586            00048         bsf     PORTB,cs          ; set the chip select line
                     00049 ;Wait the required 5mS for the write cycle timer Twc
0013 0999            00050         call    delay             ; call the delay subroutine
00051 ;Send the write enable sequence (WREN)
```


© 1997 Microchip Technology Inc.

```
0014 0486           00052         bcf     PORTB,cs        ; clear the chip select (active)
0015 0C06           00053         movlw   0x06            ; load WREN sequence
0016 002C           00054         movwf   outbyte         ; store in RAM location outbyte
0017 0955           00055         call    output          ; call the SPI output routine
0018 0586           00056         bsf     PORTB,cs        ; set the chip select line
                    00057 ;Send the read status register sequence (RDSR)
0019 0486           00058         bcf     PORTB,cs        ; clear the chip select line
001A 0C05           00059         movlw   0x05            ; load RDSR sequence
001B 002C           00060         movwf   outbyte         ; store in RAM location outbyte
001C 0955           00061         call    output          ; call the SPI output routine
001D 0955           00062         call    output          ; read the data in status reg.
001E 0586           00063         bsf     PORTB,cs        ; set the chip select line
                    00064 ;Send the write sequence (WRITE)
001F 0486           00065         bcf     PORTB,cs        ; clear the chip select line
0020 0C02           00066         movlw   0x02            ; load WRITE sequence
0021 002C           00067         movwf   outbyte         ; store in RAM location outbyte
0022 0955           00068         call    output          ; call the SPI output routine
                    00069 ;****Comment out for use with 25xx010, 25xx020 or 25xx040********
0023 0C00           00070         movlw   0x00            ; load high address byte
0024 002C           00071         movwf   outbyte         ; store in RAM location outbyte
0025 0955           00072         call    output          ; call the SPI output routine
                    00073 ;*****************************************************************
0026 0C55           00074         movlw   0x55            ; load 55 as address into w
0027 002C           00075         movwf   outbyte         ; store in RAM location outbyte
0028 0955           00076         call    output          ; call the SPI output routine
0029 0CAA           00077         movlw   0xAA            ; load data AA into w
002A 002C           00078         movwf   outbyte         ; store in RAM location outbyte
002B 0955           00079         call    output          ; call the SPI output routine
002C 0586           00080         bsf     PORTB,cs        ; set the chip select line
                    00081 ;Perform data polling (RDSR bit 0)
002D 0486           00082         bcf     PORTB,cs        ; clear the chip select line
002E 0C05           00083         movlw   0x05            ; load RDSR sequence
002F 002C           00084         movwf   outbyte         ; store in RAM location outbyte
0030 0955           00085         call    output          ; call the SPI output routine
                    00086 ;
0031 0C30           00087         movlw   0x30            ; give the spi device time to
0032 002B           00088         movwf   count           ; set the WIP bit in the
0033 02EB           00089 wait    decfsz  count,1         ; status register before coming
0034 0A33           00090         goto    wait            ; back and doing data polling
                    00091 ;
0035 0955           00092 polling call    output          ; read the data in status reg.
0036 060F           00093         btfsc   rxdata,0        ; test the WIP bit in status reg.
0037 0A35           00094         goto    polling         ; WIP clear, loop until WIP set
0038 0586           00095         bsf     PORTB,cs        ; set the chip select line
                    00096 ;Send read sequence (READ), read address 0x55
0039 0486           00097         bcf     PORTB,cs        ; clear the chip select line
003A 0C03           00098         movlw   0x03            ; load READ sequence
003B 002C           00099         movwf   outbyte         ; store in RAM location outbyte
003C 0955           00100         call    output          ; call the SPI output routine
                    00101 ;****Comment out for use with 25xx010, 25xx020 or 25xx040********
003D 0C00           00102         movlw   0x00            ; load high address byte
003E 002C           00103         movwf   outbyte         ; store in RAM location outbyte
003F 0955           00104         call    output          ; call the SPI output routine
                    00105 ;*****************************************************************
0040 0C55           00106         movlw   0x55            ; move the address into w
0041 002C           00107         movwf   outbyte         ; store in RAM location outbyte
0042 0955           00108         call    output          ; call the SPI output routine
0043 0955           00109         call    output          ; call output to read 1 byte
0044 0586           00110         bsf     PORTB,cs        ; set the chip select line
                    00111 ;Send the write enable sequence (WREN)
0045 0486           00112         bcf     PORTB,cs        ; clear the chip select (active)
0046 0C06           00113         movlw   0x06            ; load WREN sequence
0047 002C           00114         movwf   outbyte         ; store in RAM location outbyte
0048 0955           00115         call    output          ; call the SPI output routine
0049 0586           00116         bsf     PORTB,cs        ; set the chip select line
                    00117 ;Send the write disable sequence (WRDI)
```


© 1997 Microchip Technology Inc.

```
004A 0486          00118          bcf     PORTB,cs        ; clear the chip select (active)
004B 0C04          00119          movlw   0x04            ; load WRDI sequence
004C 002C          00120          movwf   outbyte         ; store in RAM location outbyte
004D 0955          00121          call    output          ; call the SPI output routine
004E 0586          00122          bsf     PORTB,cs        ; set the chip select line
                   00123 ;Send the read status register sequence (RDSR)
004F 0486          00124          bcf     PORTB,cs        ; clear the chip select line
0050 0C05          00125          movlw   0x05            ; load RDSR sequence
0051 002C          00126          movwf   outbyte         ; store in RAM location outbyte
0052 0955          00127          call    output          ; call the SPI output routine
0053 0955          00128          call    output          ; read the data in status reg.
                   00129 ;*******Go back to main routine*********************************
0054 0A01          00130          goto    start
                   00131
                   00132 ;******************SPI output subroutine**********************
0055 04C6          00133 output   bcf     PORTB,sck       ; clear the clock line
0056 04E6          00134          bcf     PORTB,si        ; clear the data line
0057 006F          00135          clrf    rxdata          ; clear the received data location
                   00136 ;
0058 06EC          00137          btfsc   outbyte,7       ; test outbyte bit 7 , skip if 0
0059 05E6          00138          bsf     PORTB,si        ; set the data line
005A 05C6          00139          bsf     PORTB,sck       ; set the clock line
005B 036F          00140          rlf     rxdata,1        ; rotate rxdata 1 to the right
005C 06A6          00141          btfsc   PORTB,so        ; test the SPI serial data out
005D 02AF          00142          incf    rxdata,1        ; incement rxdata lsb if set
005E 04C6          00143          bcf     PORTB,sck       ; clear the clock line
005F 04E6          00144          bcf     PORTB,si        ; clear the data line
                   00145 ;
0060 06CC          00146          btfsc   outbyte,6       ; test outbyte bit 6 , skip if 0
0061 05E6          00147          bsf     PORTB,si        ; set the data line
0062 05C6          00148          bsf     PORTB,sck       ; set the clock line
0063 036F          00149          rlf     rxdata,1        ; rotate rxdata 1 to the right
0064 06A6          00150          btfsc   PORTB,so        ; test the SPI serial data out
0065 02AF          00151          incf    rxdata,1        ; incement rxdata lsb if set
0066 04C6          00152          bcf     PORTB,sck       ; clear the clock line
0067 04E6          00153          bcf     PORTB,si        ; clear the data line
                   00154 ;
0068 06AC          00155          btfsc   outbyte,5       ; test outbyte bit 5 , skip if 0
0069 05E6          00156          bsf     PORTB,si        ; set the data line
006A 05C6          00157          bsf     PORTB,sck       ; set the clock line
006B 036F          00158          rlf     rxdata,1        ; rotate rxdata 1 to the right
006C 06A6          00159          btfsc   PORTB,so        ; test the SPI serial data out
006D 02AF          00160          incf    rxdata,1        ; incement rxdata lsb if set
006E 04C6          00161          bcf     PORTB,sck       ; clear the clock line
006F 04E6          00162          bcf     PORTB,si        ; clear the data line
                   00163 ;
0070 068C          00164          btfsc   outbyte,4       ; test outbyte bit 4 , skip if 0
0071 05E6          00165          bsf     PORTB,si        ; set the data line
0072 05C6          00166          bsf     PORTB,sck       ; set the clock line
0073 036F          00167          rlf     rxdata,1        ; rotate rxdata 1 to the right
0074 06A6          00168          btfsc   PORTB,so        ; test the SPI serial data out
0075 02AF          00169          incf    rxdata,1        ; incement rxdata lsb if set
0076 04C6          00170          bcf     PORTB,sck       ; clear the clock line
0077 04E6          00171          bcf     PORTB,si        ; clear the data line
                   00172 ;
0078 066C          00173          btfsc   outbyte,3       ; test outbyte bit 3 , skip if 0
0079 05E6          00174          bsf     PORTB,si        ; set the data line
007A 05C6          00175          bsf     PORTB,sck       ; set the clock line
007B 036F          00176          rlf     rxdata,1        ; rotate rxdata 1 to the right
007C 06A6          00177          btfsc   PORTB,so        ; test the SPI serial data out
007D 02AF          00178          incf    rxdata,1        ; incement rxdata lsb if set
007E 04C6          00179          bcf     PORTB,sck       ; clear the clock line
007F 04E6          00180          bcf     PORTB,si        ; clear the data line
                   00181 ;
0080 064C          00182          btfsc   outbyte,2       ; test outbyte bit 2 , skip if 0
0081 05E6          00183          bsf     PORTB,si        ; set the data line
```


© 1997 Microchip Technology Inc.

```
0082 05C6           00184         bsf     PORTB,sck       ; set the clock line
0083 036F           00185         rlf     rxdata,1        ; rotate rxdata 1 to the right
0084 06A6           00186         btfsc   PORTB,so        ; test the SPI serial data out
0085 02AF           00187         incf    rxdata,1        ; incement rxdata lsb if set
0086 04C6           00188         bcf     PORTB,sck       ; clear the clock line
0087 04E6           00189         bcf     PORTB,si        ; clear the data line
                    00190 ;
0088 062C           00191         btfsc   outbyte,1       ; test outbyte bit 1 , skip if 0
0089 05E6           00192         bsf     PORTB,si        ; set the data line
008A 05C6           00193         bsf     PORTB,sck       ; set the clock line
008B 036F           00194         rlf     rxdata,1        ; rotate rxdata 1 to the right
008C 06A6           00195         btfsc   PORTB,so        ; test the SPI serial data out
008D 02AF           00196         incf    rxdata,1        ; incement rxdata lsb if set
008E 04C6           00197         bcf     PORTB,sck       ; clear the clock line
008F 04E6           00198         bcf     PORTB,si        ; clear the data line
                    00199
0090 060C           00200         btfsc   outbyte,0       ; test outbyte bit 0 , skip if 0
0091 05E6           00201         bsf     PORTB,si        ; set the data line
0092 05C6           00202         bsf     PORTB,sck       ; set the clock line
0093 036F           00203         rlf     rxdata,1        ; rotate rxdata 1 to the right
0094 06A6           00204         btfsc   PORTB,so        ; test the SPI serial data out
0095 02AF           00205         incf    rxdata,1        ; incement rxdata lsb if set
0096 04C6           00206         bcf     PORTB,sck       ; clear the clock line
0097 04E6           00207         bcf     PORTB,si        ; clear the data line
                    00208 ;
0098 0800           00209         retlw   0               ; return from subroutine
                    00210
                    00211 ; 250 x 400nS x 50 = 5mS (plus overhead)
0099 0C32           00212 delay   movlw   0x32            ; move 50 decimal into w
009A 002D           00213         movwf   temp1           ; move 50 decimal into temp1
009B 0CFA           00214 dec1    movlw   0xFA            ; move 250 decimal into w
009C 002E           00215         movwf   temp2           ; move 250 decimal into temp2
009D 02EE           00216 dec2    decfsz  temp2,1         ; decrement temp2, skip if zero
009E 0A9D           00217         goto    dec2            ; goto decrement 2 if not zero
009F 02ED           00218         decfsz  temp1,1         ; decrement temp1, skip if zero
00A0 0A9B           00219         goto    dec1            ; goto decrement 1 if not zero
00A1 0800           00220         retlw   0               ; return both locations = 0
                    00221
                    00222
                    00223         END
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0040 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX
0080 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XX-------------- ----------------
01C0 : ---------------- ---------------- ---------------- ---------------X

All other memory blocks unused.

Program Memory Words Used:   163
Program Memory Words Free:   349


Errors   :     0
Warnings :     0 reported,     0 suppressed
Messages :     0 reported,     0 suppressed
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

SECTION 8
QUALITY AND RELIABILITY

Improving the Susceptibility of an Application to ESD - AN595 8-1
Plastic Packaging and the Effects of Surface Mount Soldering Techniques - AN598 8-19
Continuous Improvement - AN603 8-27

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

MICROCHIP

AN595

Improving the Susceptibility of an Application to ESD

Author: *David Wilkie*
Microchip Technology

INDUCED LATCH-UP

All semiconductor devices are sensitive to electrostatic discharge (ESD) damage to varying degrees. This is true whether they are soldered to a PC board in an application, or whether they are unattached in the shipping or application assembly process. Good handling techniques such as groundstraps, static free work stations and ionizers can reduce the risk of static build up during assembly. Often more attention is paid to reducing ESD during assembly than is paid to reducing ESD risk during the lifetime of the application.

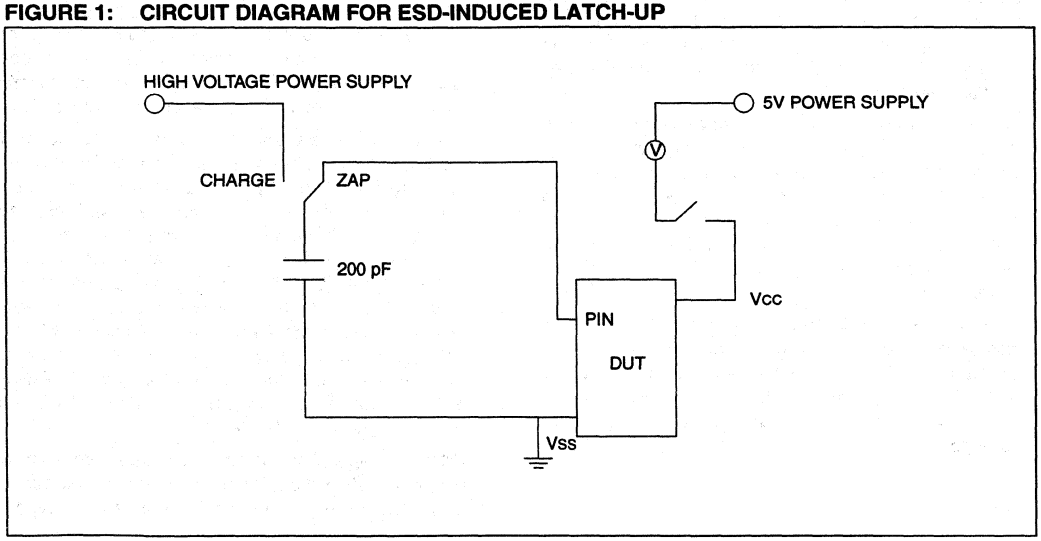
When a device is installed in an application, it is still susceptible to damage due to ESD. This can take on a different form when the application is powered up and running. If power is supplied to a CMOS device such as a memory product or a microcontroller when an ESD event occurs, the device can be triggered into a "latch-up" condition. This is a high current mode where internal circuitry can be disturbed into making a short circuit (or a circuit with very low resistance) between power (Vcc) and ground (Vss) on a device. This condition is self-sustaining; it does not require subsequent ESD events to continue the latch-up condition.

This short circuit will tend to reduce the voltage level on the application (particularly if the application is battery powered) and will do a great deal of damage to the device which has latched-up. The only way to halt this condition is to remove power from the device.

Microchip uses careful design practices to reduce the susceptibility of all products (microcontrollers or memories) to ESD events. However, the protection level varies from pin to pin, reflecting the different functions of each pin. Certain types of pins (notably supply pins) are much more susceptible to latch-up caused by ESD pulses than other pins. This is due to the different design and layout considerations that reduce the effectiveness of ESD protection.

There is a great deal that the system designer can do to improve (by up to an order of magnitude) the level of protection of a device from latch-up inducing ESD events. This tutorial is intended as a guide for helping designers choose protection. This type of protection is application dependent, so consideration should be made of the type of environment that the application, or the device, will be in.

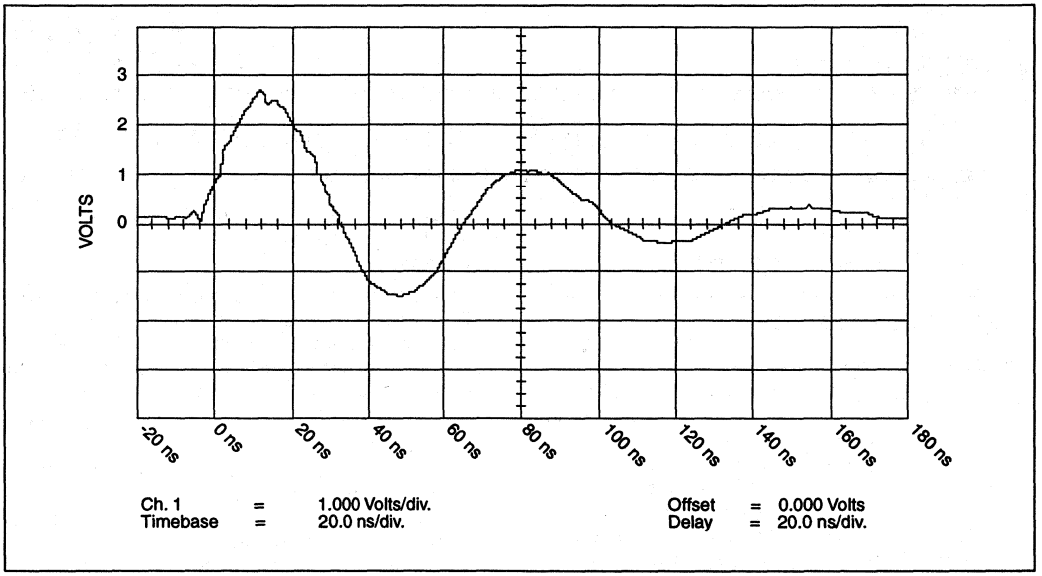
FIGURE 1: CIRCUIT DIAGRAM FOR ESD-INDUCED LATCH-UP



8

© 1997 Microchip Technology Inc.

FIGURE 2: WAVEFORM USED TO SIMULATE ESD EVENT



ELECTROSTATIC DISCHARGE

Electrostatic discharges can come from a variety of sources. The traditional ESD pulse is caused by a body at very high potential coming into contact, or near contact, with a grounded object. This could be a human body, a piece of electrical equipment, or even a piece of furniture.

In a dry environment, where static dissipation is low, a human body can develop tens of thousands of volts of potential. Almost everyone has experienced the shock of walking across a new carpet and touching a door handle. An audible snap can be heard when the potential difference between the person and the door handle is around 5,000V.

Any piece of equipment which has metal components moving against other metal components can develop a charge. An automated device handler will generally have a metal tray where devices move around, with pins in occasional contact with the tray. Static build up can reach hundreds of volts and can damage the devices in the same way as an ungrounded human handler can.

Incorrectly placed ionizers, meant to improve static dissipation, can build large potentials on office or laboratory furniture. Any person touching such a piece of furniture might feel a shock. Any devices being placed on a table with a large potential can suffer damage.

All these situations can be avoided by careful handling procedures. Groundstraps for manual handling of devices is essential, as are grounded tables and work surfaces with anti-static surfaces such as metal or specially designed plastic mats. Equipment can be carefully grounded wherever the potential exists for static build up.

ELECTROSTATIC DISCHARGE IN A POWERED APPLICATION

Once the device has been mounted to a board or installed in an application, most types of ESD events will no longer occur. For example, unless the PC board is out in the open, it is very unlikely to be touched by the user, and so a direct ESD pulse will not be a concern.

However, there are several sources of indirect ESD pulses, or noise pulses which are very similar in nature and magnitude to ESD pulses. ESD pulses can induce currents in nearby wiring. So, for example, if the user creates an ESD event on the casing of an application, the magnetic field of the pulse can induce currents inside the casing, in the wiring of an application. This electromagnetic interference can occasionally be seen in other ways, such as a noisy TV picture when a vacuum cleaner is being used, or a "click" heard on the radio when a light switch is turned on. This type of event is often called Radio Frequency Interference (RFI) or Electromagnetic Interference (EMI).

Often the application itself can induce noise spikes during operation. If one component in that application is a high speed, high current transistor or other type of

© 1997 Microchip Technology Inc.

switch, the sudden change in current can induce a noise spike which could be seen by that component, or other components in the system. This switching noise is endemic in systems with metal wires, and can not be removed completely. Large magnitude pulses of switching noise can induce latchup on sensitive CMOS devices.

The effects of switching noise or EMI can be catastrophic to an application with sensitive components. Even components which are not particularly sensitive or already have some built-in protection (such as Microchip Technology Inc.'s products) can still be latched-up by noise pulses if they are of a large enough magnitude.

PROTECTING AGAINST ESD PULSES

Basic protection can be provided by a simple decoupling capacitor, placed as close as possible to the power and ground pins of components. Each component should have its own capacitor; simply decoupling the whole application at the supply points will not be sufficient if a component in the application is producing noise.

If the value of the capacitor is chosen to match the device, then non-supply pins can also benefit from a decoupling capacitor between power and ground. A single capacitor can not filter out all the frequencies associated with a noise spike, but it can still offer very effective and low cost protection.

We used a simple test circuit to accurately model an ESD-induced noise spike. The test circuit is shown in Figure 1. A 200 pF capacitor was charged to various voltages and then switched to discharge directly into the device being tested. The waveform is a high frequency (≈14 MHz) and short period (≈100 ns) decaying oscillation. The resulting waveform, measured through a Tektronix CT-1 current probe with a 10X attenuator, is shown in Figure 2. The testing was conducted at 25°C.

The period of the oscillation is governed by the relationship:

$$\omega = \sqrt{\frac{1}{LC} - \frac{R^2}{4L^2}}$$

where L, C, and R are the inductance, capacitance, and resistance of the oscillating circuit, and w is the frequency of the oscillation. Since R and L are very small, and C is 200pF, the oscillation period is very fast.

Two devices were tested. A typical serial EEPROM, the 24LC04B, was tested on all function and supply pins. The results, shown in graphical form (Figure 3 through Figure 7) show that, even with a capacitor placed between power (Vcc) and ground (Vss) other pins, such as SDA can have increased protection. From the figures it is clear that the best all-round protection can be obtained from a 10,000 pF capacitor.

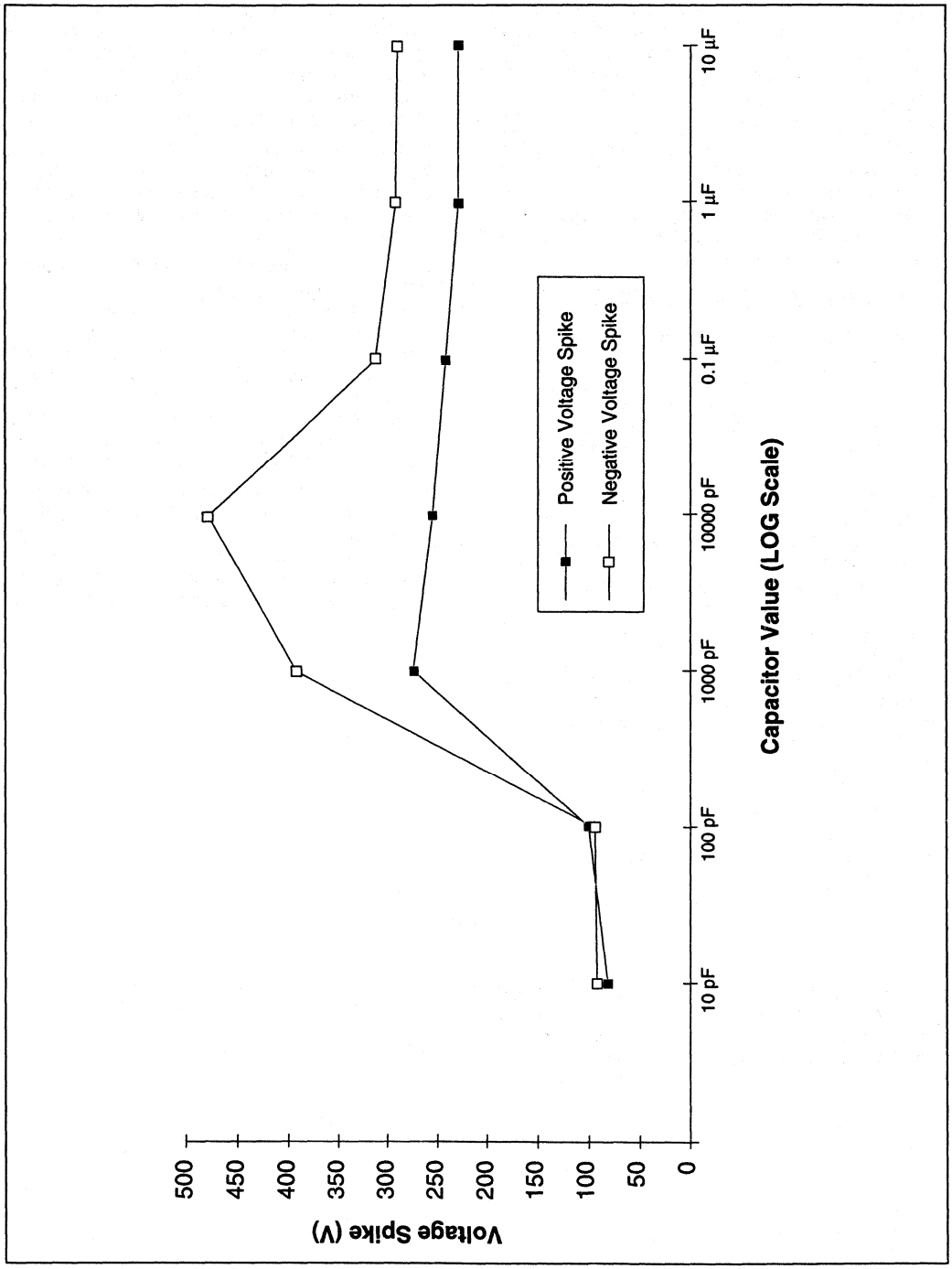
Particular applications may be different. For example, the designer may know that there is a better chance for a noise spike on the SDA signal, and so may want to use a 1,000 pF capacitor to improve the protection level of SDA.

A PIC16C54 microcontroller in XT mode was tested on all functional and supply pins. The results are shown in Figure 8 through Figure 17. From these results, it is clear that the particular application environment will be much more important in determining which capacitor value to use. Some pins, such as $\overline{\text{MCLR}}$, do not respond at all to a decoupling capacitor. Others, such as the RA ports, respond strongly to a capacitor. Note that for the RA and RB graphs, the pin with the lowest latch-up threshold was used. All other pins are higher.

It is important for the designer to be aware of the potential problems associated with noise in a powered application. A combination of using sound design techniques to reduce causes of noise and an awareness of the protection levels of the components being used can make the problems of ESD manageable.

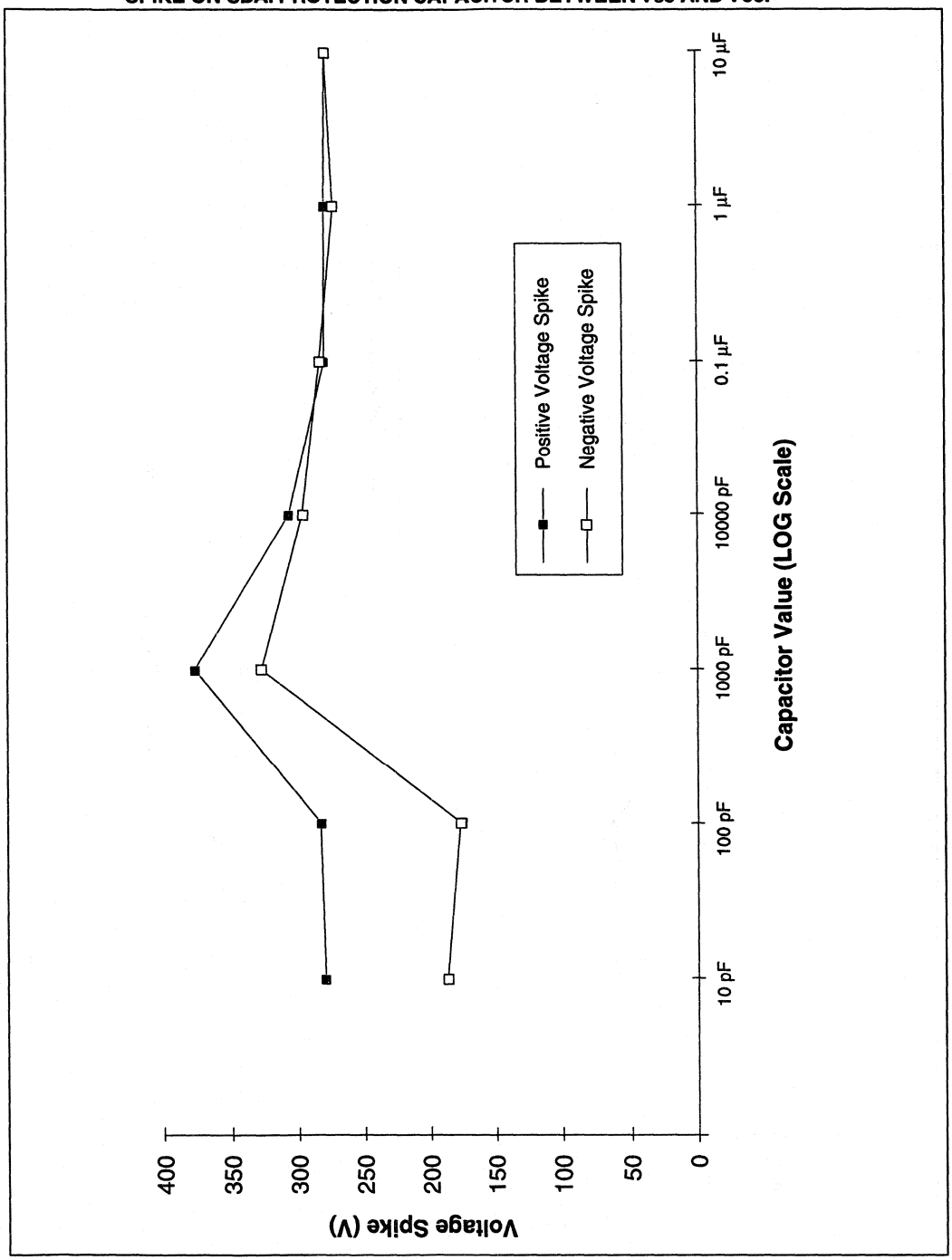
© 1997 Microchip Technology Inc.

FIGURE 3: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON Vss. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



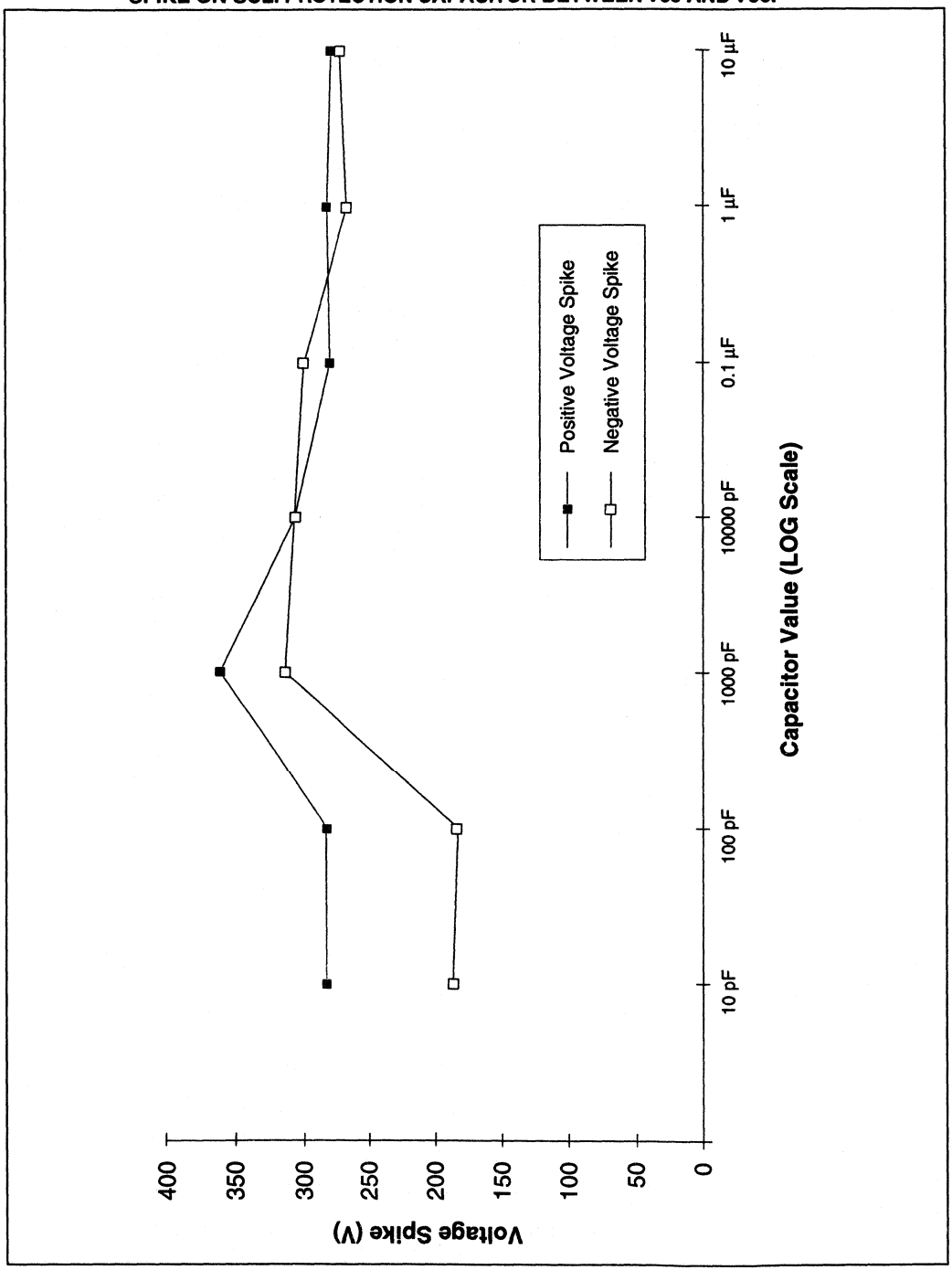
© 1997 Microchip Technology Inc.

FIGURE 4: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON SDA. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



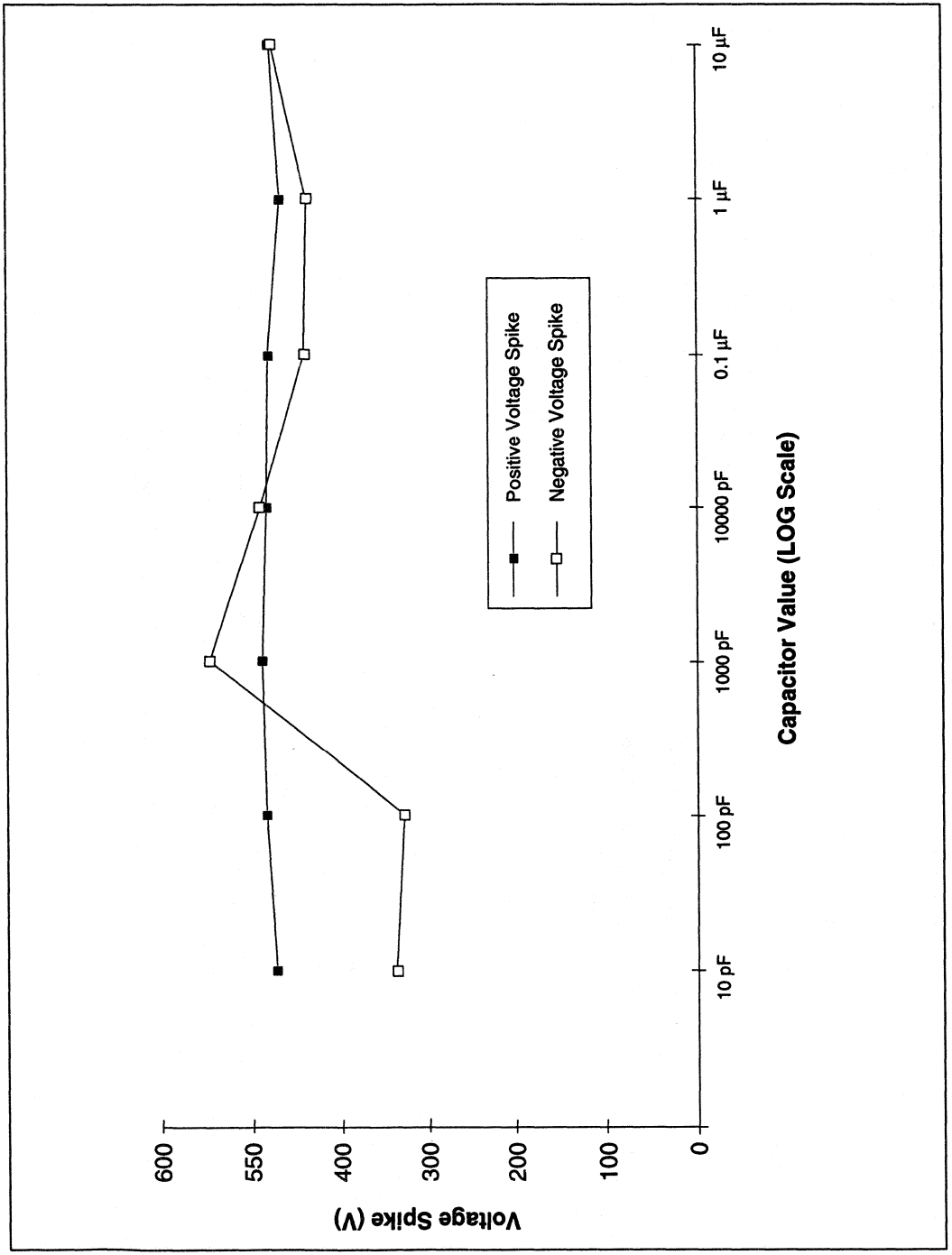
8

FIGURE 5: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON SCL. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



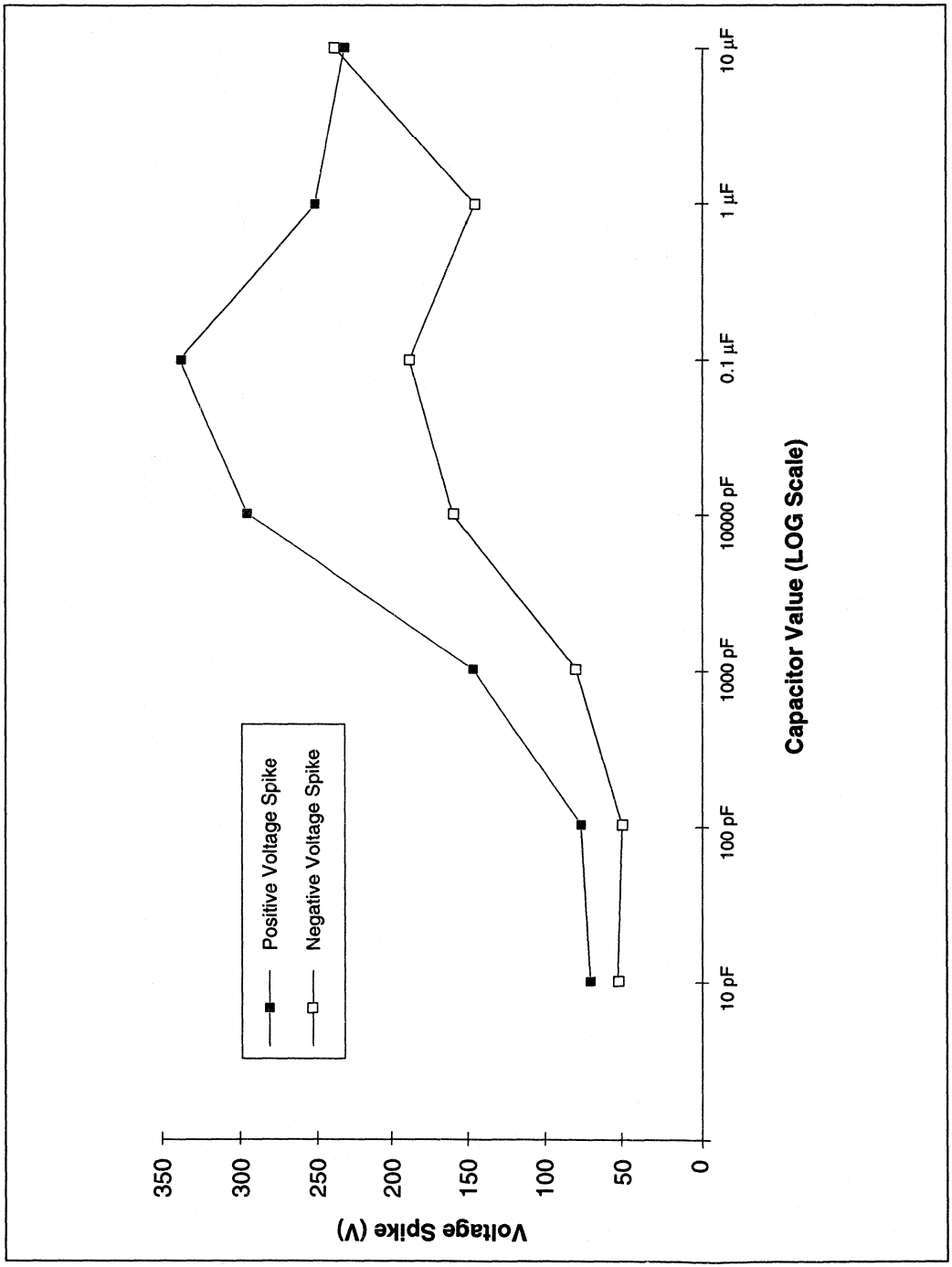
© 1997 Microchip Technology Inc.

FIGURE 6: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON WP. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



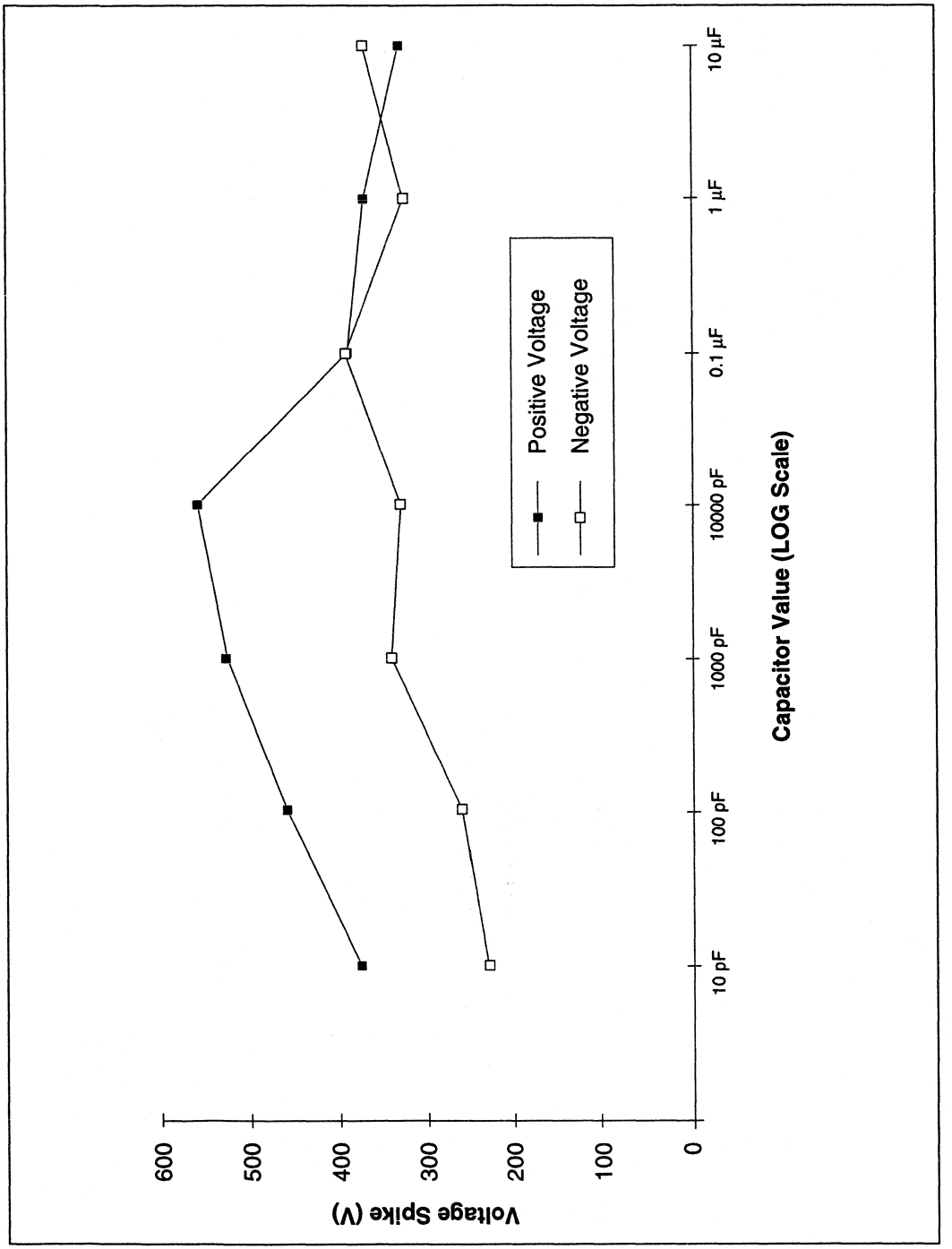
© 1997 Microchip Technology Inc.

FIGURE 7: 24LC04B PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON Vcc. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



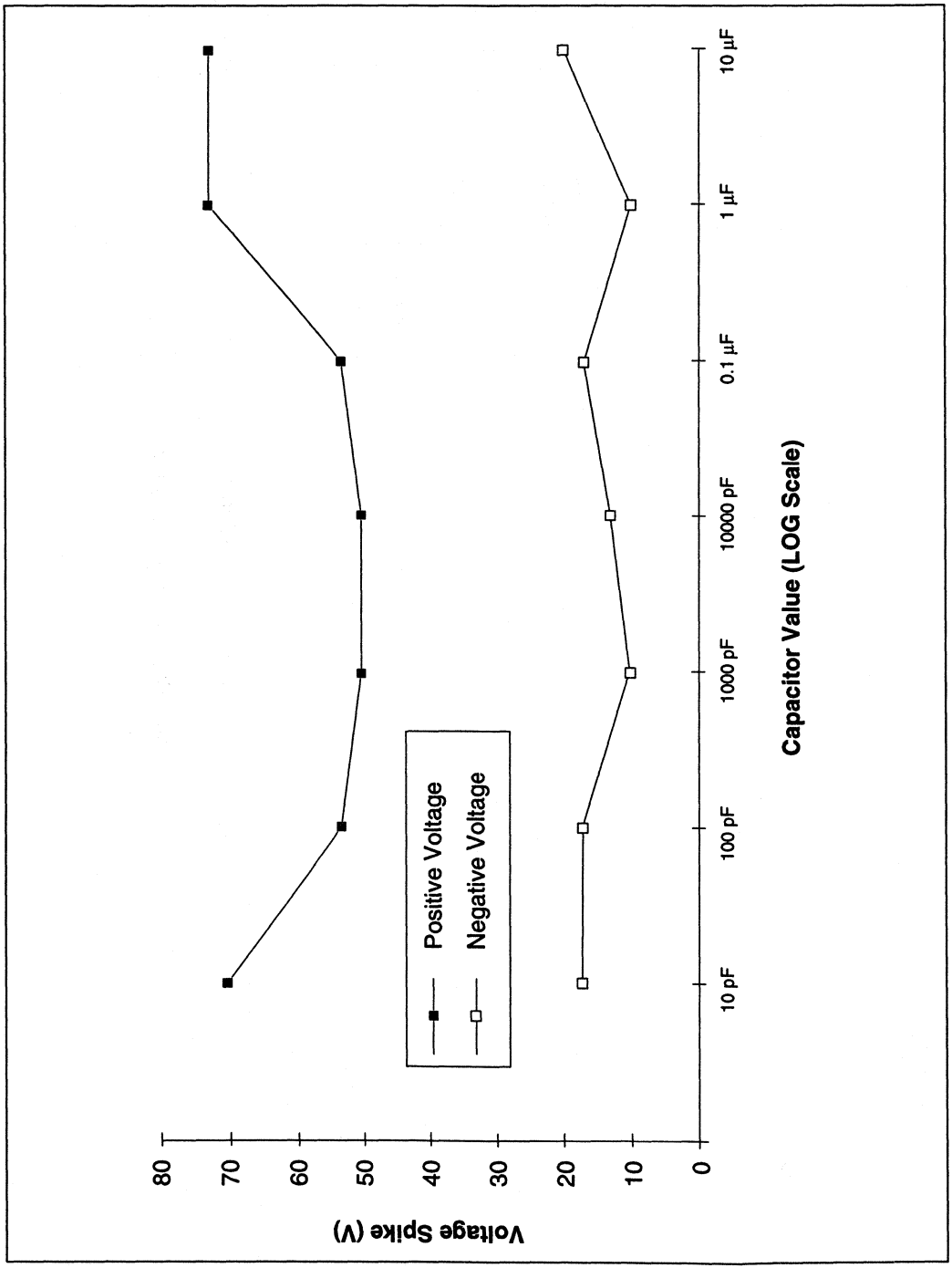
© 1997 Microchip Technology Inc.

FIGURE 8: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON RTCC. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



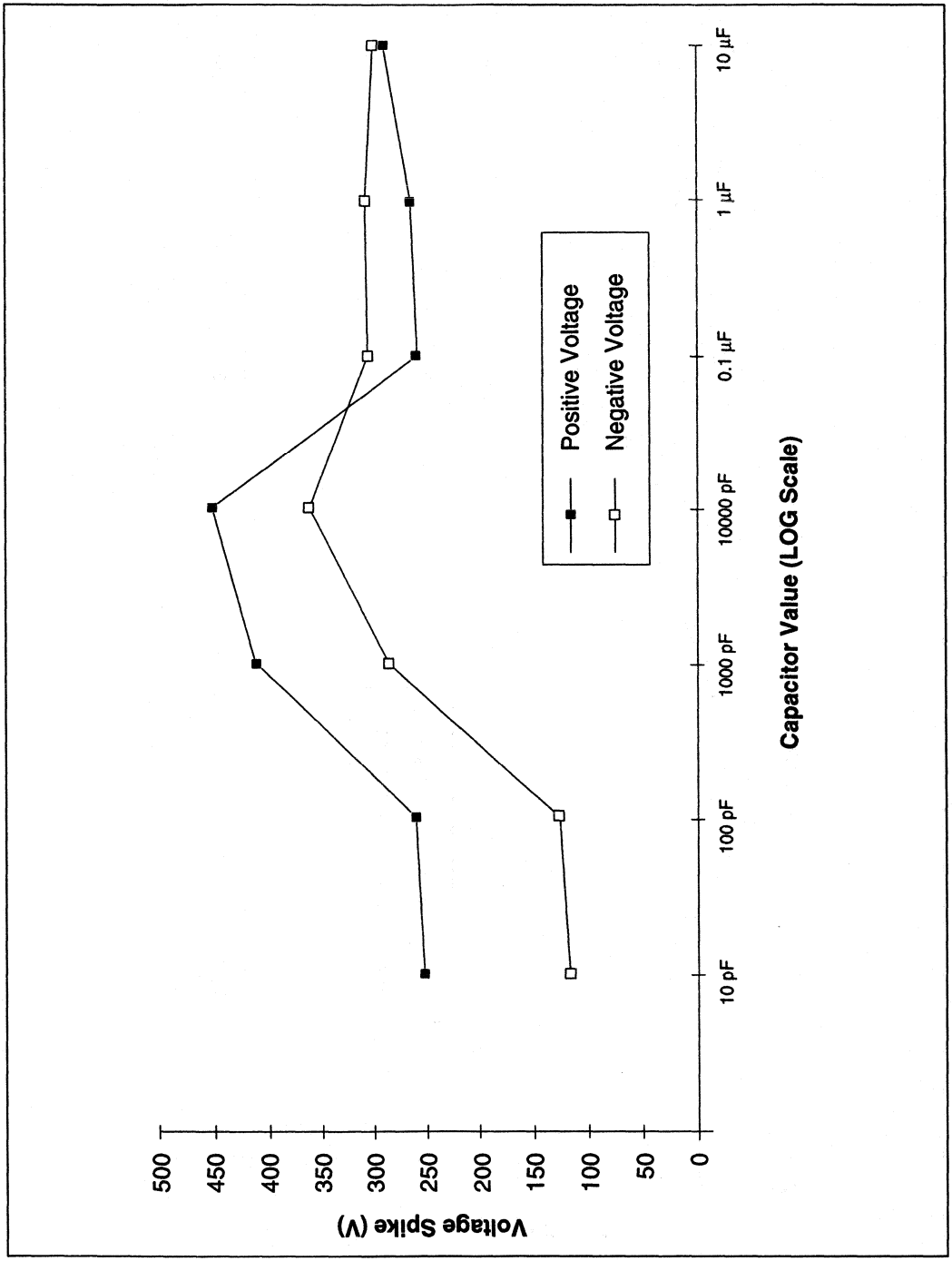
© 1997 Microchip Technology Inc.

FIGURE 9: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON $\overline{MCLR}$. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



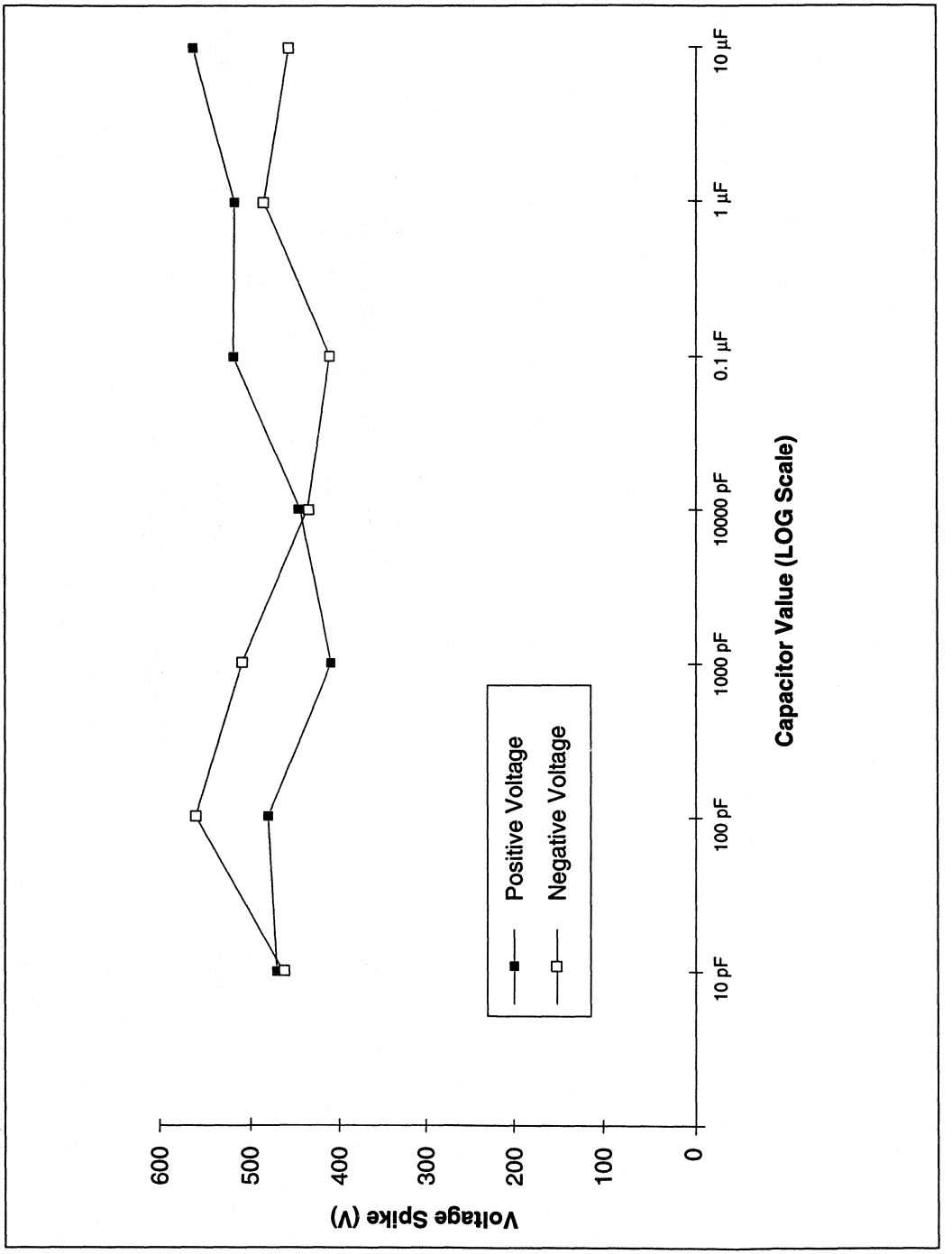
© 1997 Microchip Technology Inc.

FIGURE 10: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON Vss. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



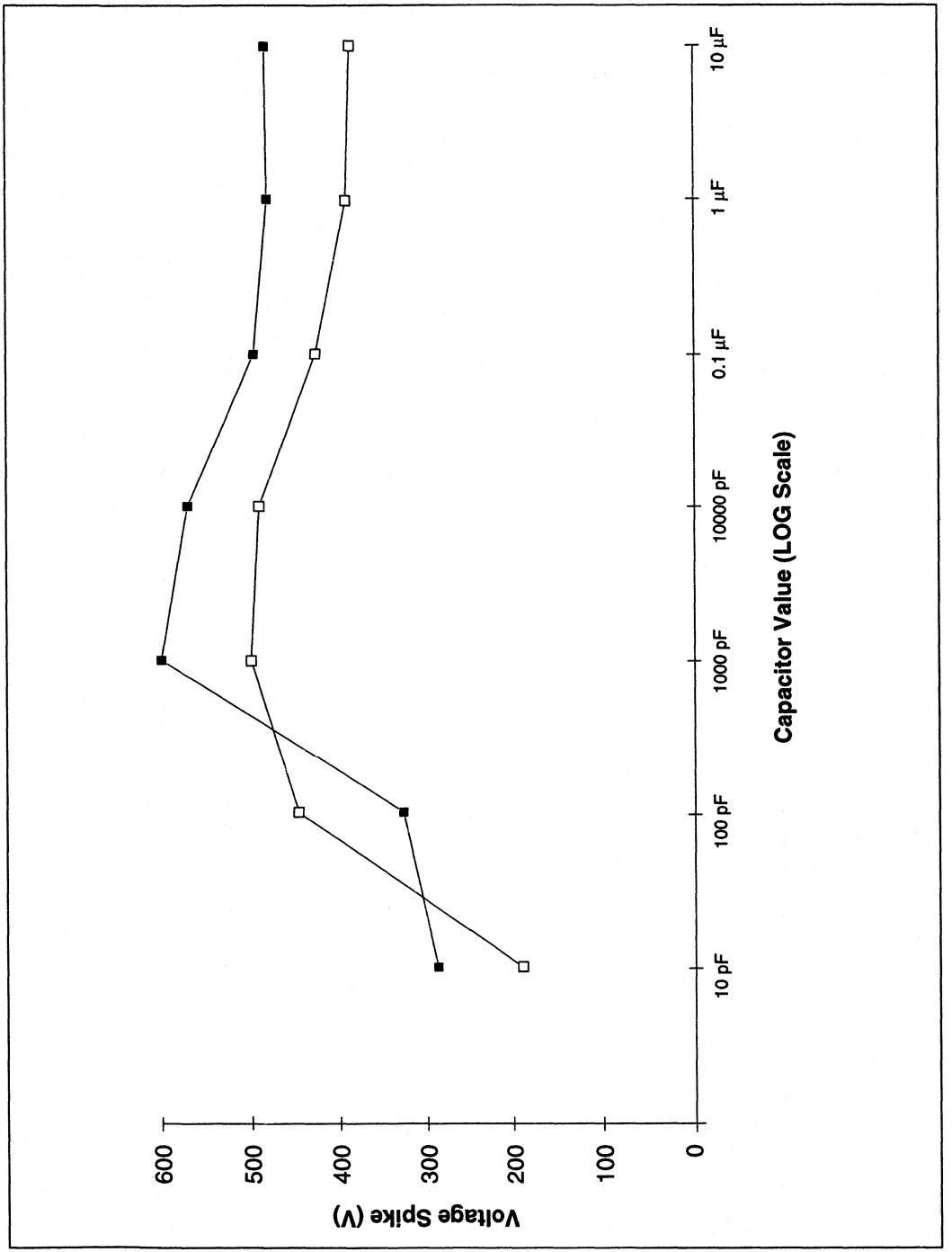
© 1997 Microchip Technology Inc.

FIGURE 11: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON RB PORTS. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



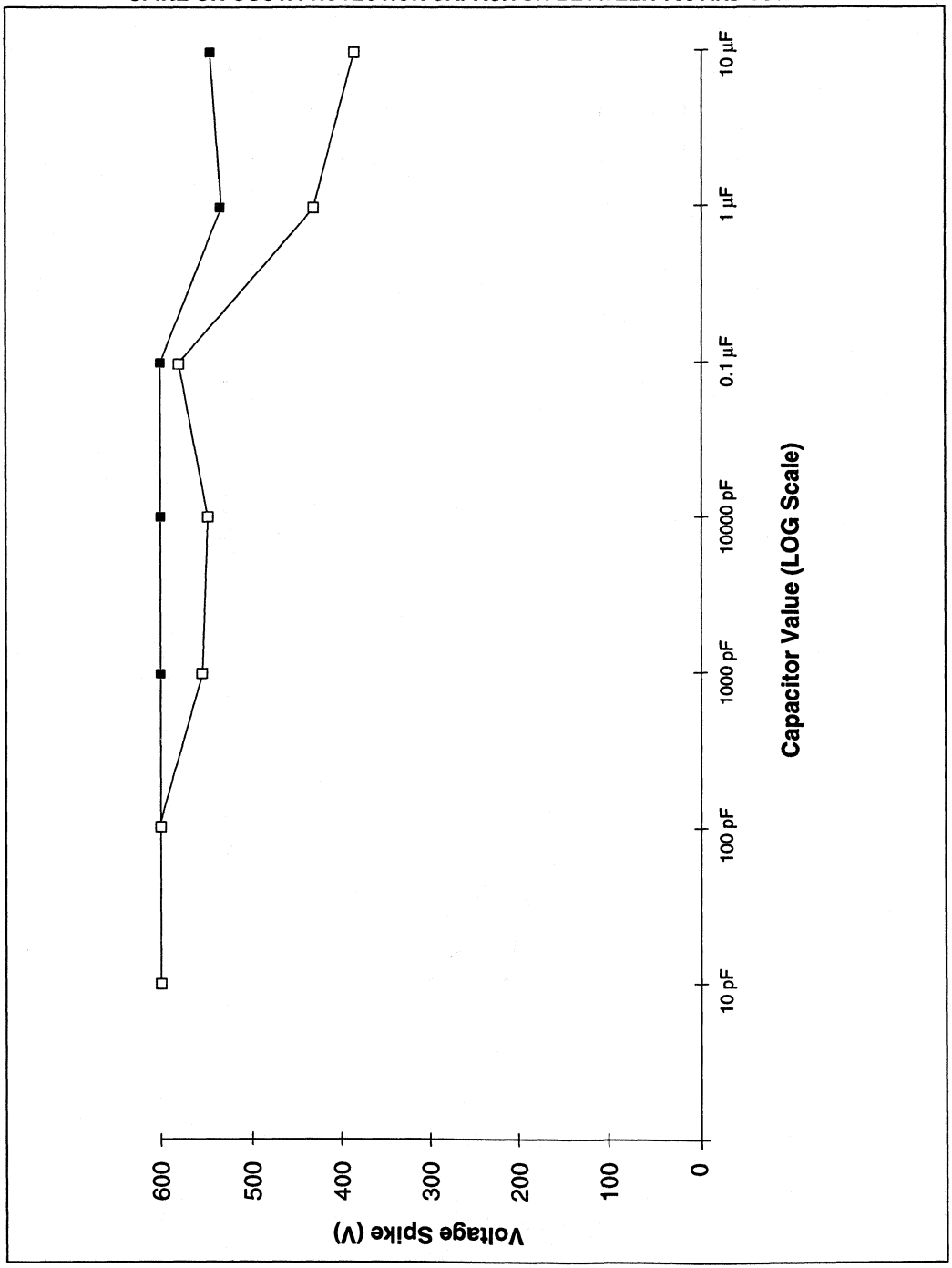
© 1997 Microchip Technology Inc.

FIGURE 12: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON RA PORTS. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



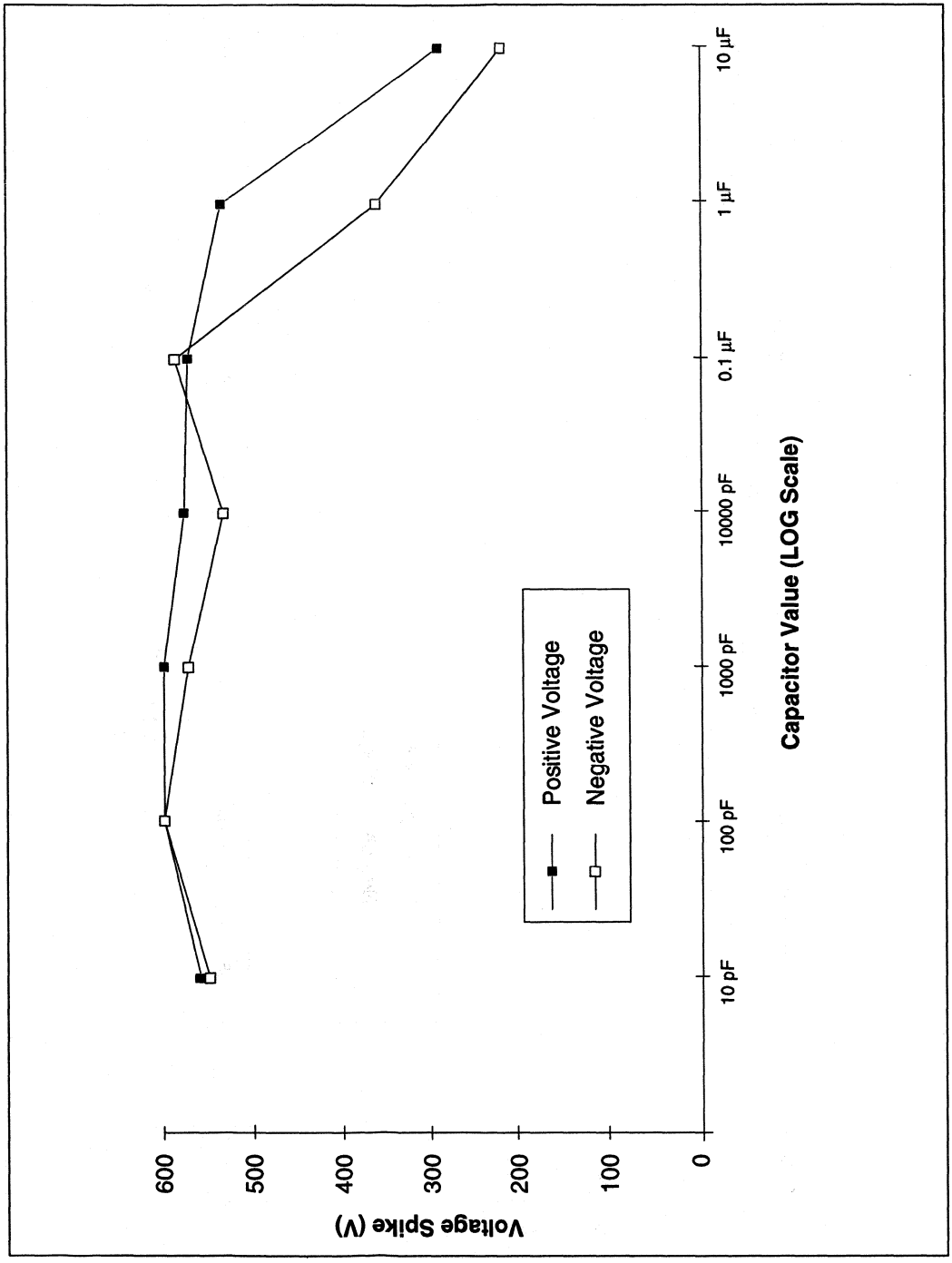
© 1997 Microchip Technology Inc.

FIGURE 13: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON OSC1. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



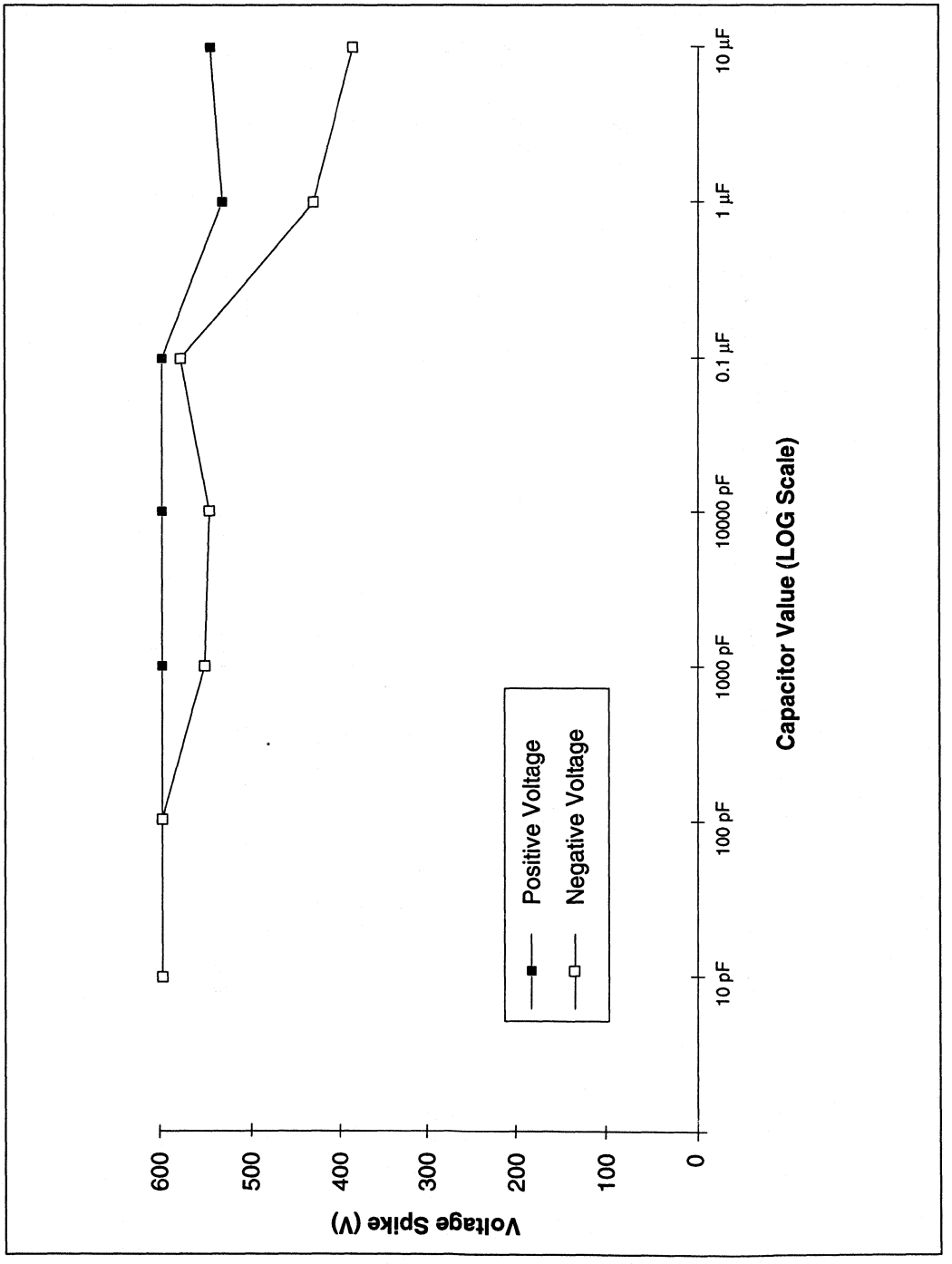
© 1997 Microchip Technology Inc.

FIGURE 14: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON OSC2. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



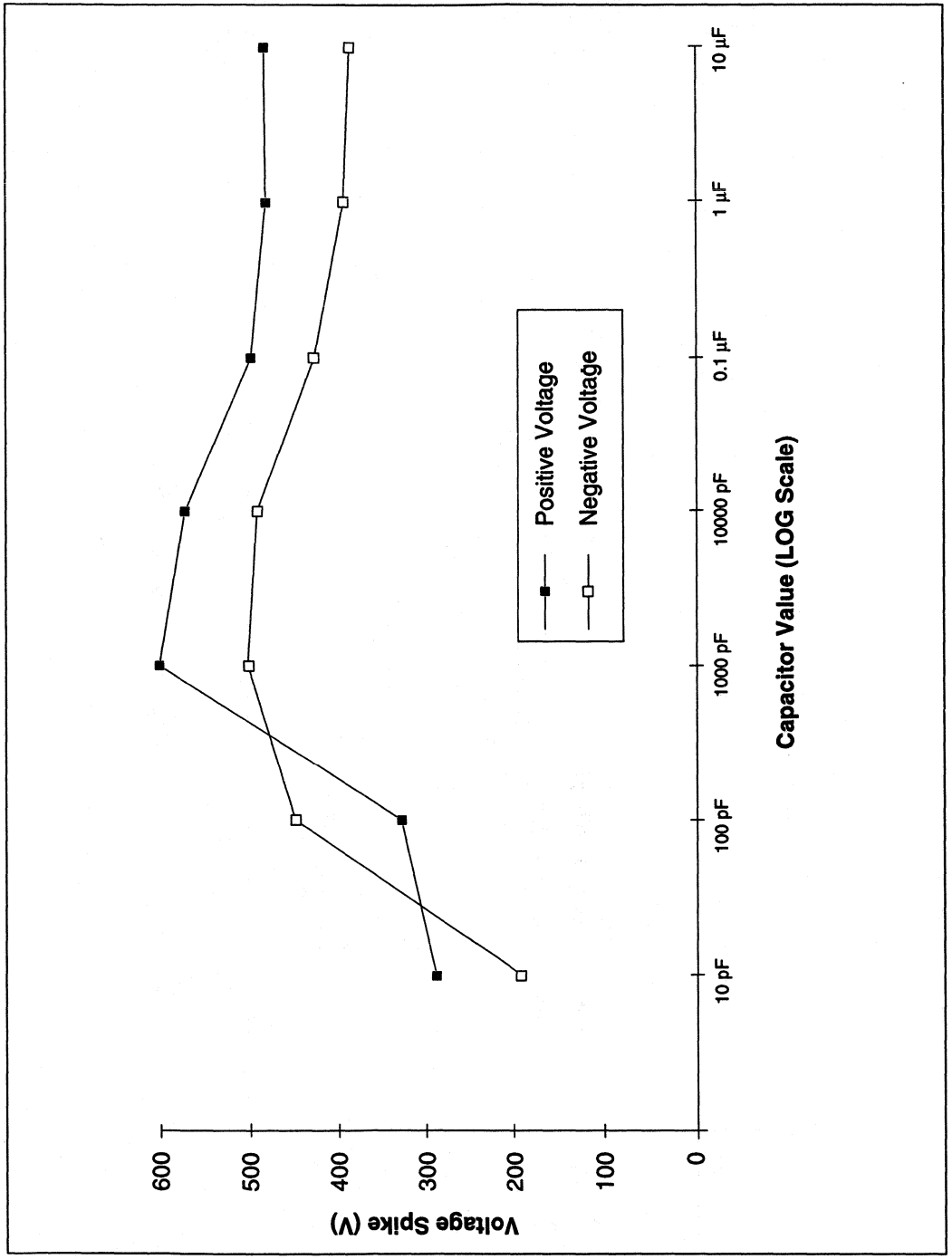
8

FIGURE 15: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON OSC1. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



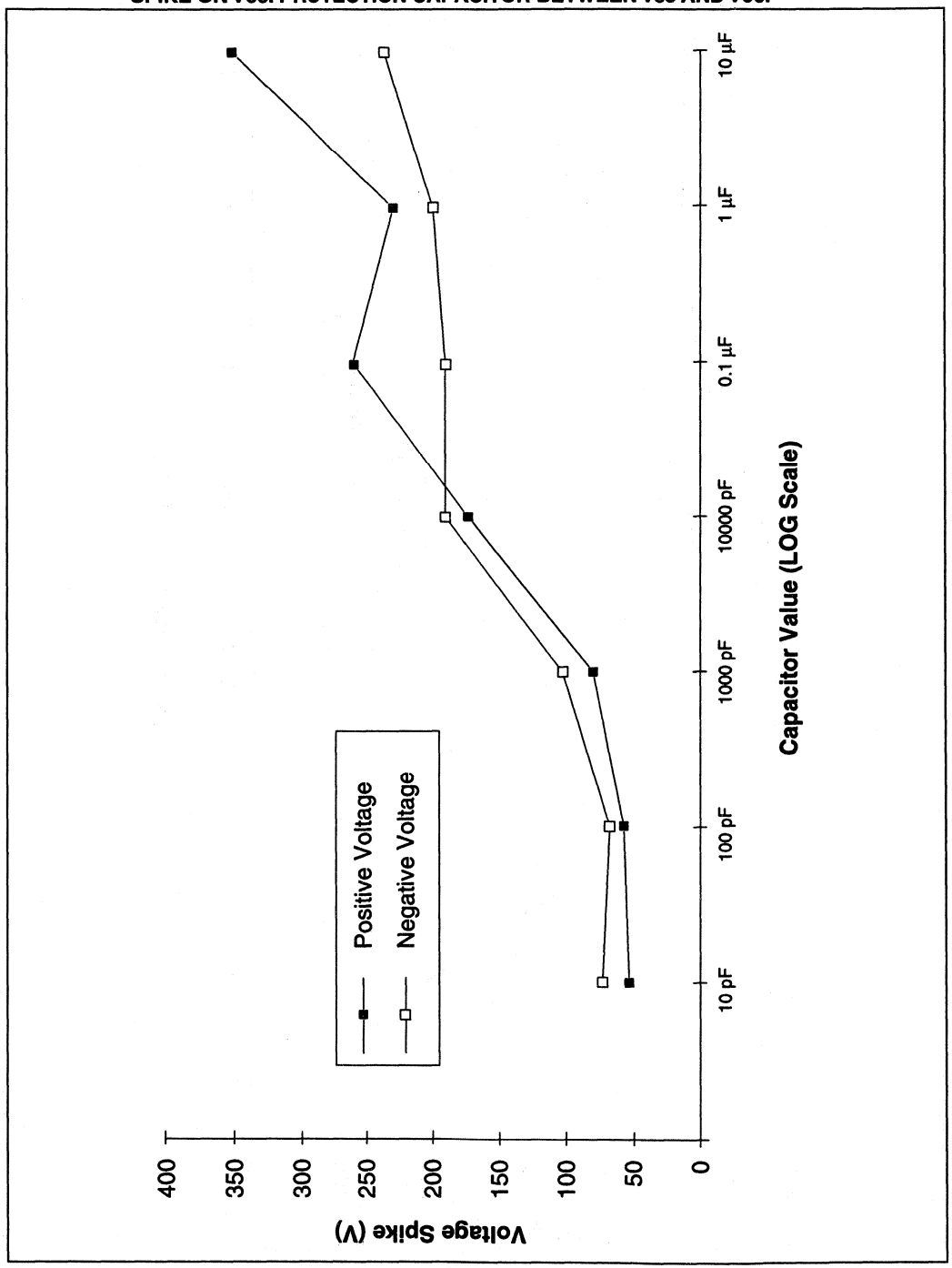
© 1997 Microchip Technology Inc.

FIGURE 16: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON RA PORTS. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



© 1997 Microchip Technology Inc.

FIGURE 17: PIC16C54-XT PROTECTION LEVELS FOR ESD-INDUCED LATCHUP WITH VOLTAGE SPIKE ON Vcc. PROTECTION CAPACITOR BETWEEN Vss AND Vcc.



© 1997 Microchip Technology Inc.

MICROCHIP

AN598

Plastic Packaging and the Effects of Surface Mount Soldering Techniques

Author: *John Barber*
Microchip Technology Inc.

PURPOSE

This application note is intended to inform and assist the customers of Microchip Technology Inc. with Surface Mount Devices (SMD's). The process of packaging a semiconductor in plastic brings to pass a somewhat unlikely marriage of different materials. In order to minimize potential adverse effects of surface mount solder techniques, it is worthwhile to understand the interaction of the package materials during the time they are subjected to thermal stress. Understanding both the limits of thermal stressing that SMD's can withstand and how those stresses interact to produce failures are crucial to successfully maintaining reliability in the finished product. A recommended Infrared (IR) solder profile is provided as a reference later.

The electronics industry has moved to smaller and thinner surface mount packaging in the progress toward miniaturization of circuits. This trend has necessitated the use of lower profile and smaller footprint packages. There has been an increase in reliability problems corresponding with the shrinking size of plastic SMD's. These problems manifest themselves in such ways as moisture sensitivity, cracked packages, open bond wires and intermittent continuity failures. Problems of this type are not present in the devices prior to assembly onto printed circuit boards but are the result of thermally induced stressing to the part during assembly or any rework such as desoldering.

WHAT CAN HAPPEN DURING THE SOLDER PROCESS

In surface mount soldering both the body and leads are intentionally heated. This direct heating of the reduced sized device package is at the heart of the problems experienced with Surface Mount Technology (SMT). Older techniques were concerned with the heating of the leads only. Some degree of heating was present in the body due to the thermal conductivity of the leadframe but this did not produce the same level of stress that devices are now subject to with SMT.

The heat from soldering causes a buildup of additional stresses within the device that were not present from the manufacturing process. Board level solder processes, such as IR reflow and Vapor phase reflow, are well-known areas where temperatures can reach levels sufficient to cause failure of the package integrity. A plastic semiconductor package forms a rigid system where the various components are locked together. Differences in the physical expansion rate of materials will result in internal package stresses because the constituent parts cannot move. When a package is heated, the stresses in the device are applied to the die in such a way that the maximum areas of stress[1] are at the corners. Forces can build to the point where the areas of adhesion between different components of the package give way causing device failure. Failure modes associated with excess stress include delamination of surfaces, fracturing of bond wires, die cracking, cratering of bond pads and package cracking.

Moisture sensitivity of plastic packages has been a concern for semiconductor manufacturers. Moisture can and will permeate any molding compound. The rate of permeation will vary with package compound thickness and type. Relative humidity will also play a role in the time required to saturate a device.

Moisture content will affect the ability of a device to withstand the stresses of surface mount soldering. If sufficient moisture is present inside of a device during soldering, high temperatures can cause steam which has the potential to crack the package. This type of damage is commonly called "pop-corning"[2].

Moisture can lead to corrosion of exposed aluminum metallization inside the device. Fortunately, a film of water is required[3] for corrosion to take place. Water vapor alone is not sufficient to produce the onset of corrosion. It is difficult to collect a film of water inside a package where there is no defect.

Chemical compatibility is important to control corrosion of aluminum (especially in the presence of moisture). Most molding compounds and die attach epoxies contain free ions that can lead to corrosion under conditions where moisture is available to support the chemical reaction(s). Careful selection and handling of materials minimize the number of chemical impurities in the device that could lead to corrosion.

© 1997 Microchip Technology Inc.

MATERIAL INTERACTIONS

To understand the significance of what is happening to a device during solder reflow, it is necessary to understand something about a few specific material properties and how those properties interact. Physical properties of interest[4] are listed for reference in Table 2. These properties will be defined as needed to explain various concepts.

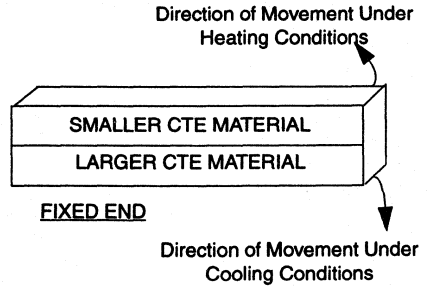
There are five major components used in a plastic package. Basic package components are: (1) epoxy molding compound, (2) leadframe made of copper or Alloy 42, (3) die attach epoxy, (4) silicon die and (5) gold bond wires. Molding compounds have several significant contributing factors that define their performance. These must be considered by the manufacturer when a selection is made. These items are Temperature of Glass Transition (Tg), Coefficient of Thermal Expansion (CTE), moisture absorption characteristics, flexural modulus and strength, and thermal conductivity. In reality, molding compound suppliers provide the test bed for development of compounds, only a few of the very large semiconductor manufacturers have published data[5] suggesting independent experimentation into this area. Leadframes used by Microchip are copper with a silver plated area for die attach and wire bonding. Die attach material is typically a silver filled epoxy. The silver is added for thermal and electrical conduction. Plastic devices have gold bond wires.

Thermally induced transients generated during surface mount soldering can have a significant impact on the reliability of plastic encapsulated devices in the field. In most cases, thermal transients below 125°C are not of sufficient magnitude to cause damage to the part. As a device experiences a significant thermal transient, such as would result from IR reflow soldering, the differing materials expand and contract at unequal rates. The CTE describes the behavior of a material as it expands or contracts when subjected to a temperature change. Materials with similar expansion coefficients will have similar thermal behavior if the phase boundaries are not approached. The CTE characterizes performance over a given temperature range. Table 2 shows that there is a change in the coefficient (ΔCE) of roughly 12.9 (16 - 3.1 = 12.9) between a copper leadframe and silicon, and a ΔCE of 4.4 (7.5 - 3.1 = 4.4) between silicon and lowest stress molding compounds available. The rate of expansion and contraction is not a simple linear relationship with temperature change. The rate can vary dramatically with the phase of the material. For example, molding compounds will greatly increase the rate of expansion when temperatures above the Tg is reached.

In general terms, Tg is the temperature where the material changes from a solid to something more like a plastic or vice versa. More precisely, the temperature of glass transition would be the temperature at which atoms, in chains of 30 to 40 atom groups, start to move. Generally, Tg is in the area of 170°C for molding compounds. Several factors affect Tg such as the basic formulation of the resin from the supplier, cure time and the temperature used in the manufacturing process. A common misconception when trying to relate to these concepts is to assume that the entire package is at the same temperature at any given moment in time. This is not the case.

To aid in understanding the effects of CTE mismatching, take, for example, a bi-metallic strip. They can be found in every automobile and are used in turn signal flasher units for controlling the flashing action of the lights used as turn indicators. Inside the flasher unit the contact arm is formed by use of a bi-metallic strip to make and break electrical contact.

The metals used in the construction of a bi-metallic strip have different CTE's and are specifically chosen to produce a desired effect. These two materials are bonded so that they move together. In the automobile example, current flowing through the bi-metallic strip causes local heating of the strip. Due to the unequal CTE's, the strip will bend away from the contact as it is heated which causes the circuit to open. When current stops flowing in the strip, heating also stops. The bending action of the strip is produced by unequal expansion on one side of the strip. As the strip cools, it will move back thus making contact again allowing current induced heating which starts the cycle over again. In similar fashion as bi-metallic strip, the die in a plastic package will be stressed when subjected to a thermal transient.



Adhesion of materials is a matter of importance since it is in areas of delamination that moisture can collect. The collecting of moisture can lead to corrosion problems.

Differing rates of expansion and contraction can make joining two materials a distinct challenge. Especially if the materials are significantly different in other characteristics that affect surface adhesion. Some materials that are not suitable for use in the packaging system require special adhesion promoting modifications of the surfaces. The topic of adhesion is beyond the scope of this work, but is an important factor and should be given careful attention[6].

© 1997 Microchip Technology Inc.

The property known as "Fracture Toughness" is the ability of the material to resist the propagation of a fracture once the defect has been initiated. Silicon has a very low fracture toughness, therefore, a fracture will readily propagate in silicon. An example of a material at the other end of the spectrum would be Gold, one of the ductile metals, which has a high fracture toughness. This property is responsible for the tendency for glass or tungsten carbide to shatter rather than simply chip or crack. In semiconductor devices, fracture toughness should be considered when cratering and die cracking are a problem.

The closest point to a zero stress state in a plastic package is at the temperature used to cure the molding compound (≅175°C). Present plastic molding compounds are thermosetting polymers. Thermosetting means that the compound sets up and becomes hard as a result of being heated. A cooling cycle after the set up period does not undo the process. The molding compound is held at an elevated temperature for the time period required to harden or cure. During the subsequent cooling, after cure, stresses are trapped in the package because of the differing CTE's. This trapping of stress results in a net compressive force, at room temperature, on the die surface in a plastic molded device. As external stresses such as thermal stress from soldering are presented, if stresses are of sufficient magnitude, the material strength will be exceeded resulting in package damage. Manufacturing processes tend to leave stresses trapped inside the devices. Thickness of the die attach material is regulated to control stressing levels due to its presence. A thin die attach will result in higher tensile stress on the die surface after die attach. The interaction of the die attach material and the molding compound result in the compressive stress on the die surface[7] after packaging. Low stress molding compounds are used on SMD's to minimize the thermal stresses generated during soldering operations.

FAILURE MODES AND MECHANISMS

Let us now review the types of damage that may be seen as a result of SMT soldering. Package damage may be manifested in several ways and may not always result in immediate device failure. Following is a list of failure types and their morphology:

1. Delamination of the molding compound along the leadframe interface and or die surface can take place. This delamination, or separation, can provide a path for moisture and contaminant ingress and pooling along the interfaces where the materials are no longer adhering to each other. This condition may lead to corrosion related problems[8].
2. Cracking of the mold compound can produce immediate failure if the crack crosses a bond wire or it can allow similar moisture effects[9] as in delamination. It can also produce intermittent contact problems.
3. Cracking of the die is generally seen as a functional failure but can be temperature sensitive if the crack is in a more benign area of the die.
4. Cratering of wire bonds is characterized as a phenomenon where portions of the silicon below the ball bond are fractured. The ball bonds pull up "plugs" or "chunks" of silicon, thereby, leaving crater shaped damaged spots in the bulk silicon below the bond pad. Cratering is a possible result of lateral stress on ball bonds. A cratering failure will typically but not always show up during electrical testing. However, it can lay dormant until another temperature excursion comes along which causes the Al metal conductor (bond pad) to open. This is a result of the soft nature of the aluminum used for interconnects in semiconductor devices. The silicon below a bond pad can be damaged without breaking contact in the aluminum. Intermittent or thermally sensitive continuity failures may be produced.
5. Moisture inside the device may collect in the die attach, along material interfaces (primarily leadframe to mold compound and leadframe to die surface) or in the molding compound. Rapid heating causes pressure build up as the moisture expands. This results in delamination and cracking of the package along with the failure modes associated with those phenomena.
6. Corrosion is heavily related to moisture effects and is intensified by delamination and cracks. The typical failure mode for corrosion is loss of continuity since the area most often affected is in the bond pads where there is no passivation layer to give additional protection to the metal.

HOW TO CORRECT THIS PROBLEM: A CASE STUDY

After the device leaves the factory there are two things that need to be done to maintain reliability.

1. The exposure to thermal stressing needs to be minimized. This is not always an easy task in light of the varied components that may go onto a single board. Some will require different conditions to solder due to there bulk size.

   Microchip carried out a partial factorial experiment on 32-lead PLCC devices with the intent of verifying an industry standard profile and providing a specification for our various customers. The devices were subjected to IR solder reflow profiles with ramp rates ranging from 1.5°C/second to 4°C/second and maximum temperatures from 220°C to 310°C (Table 1).

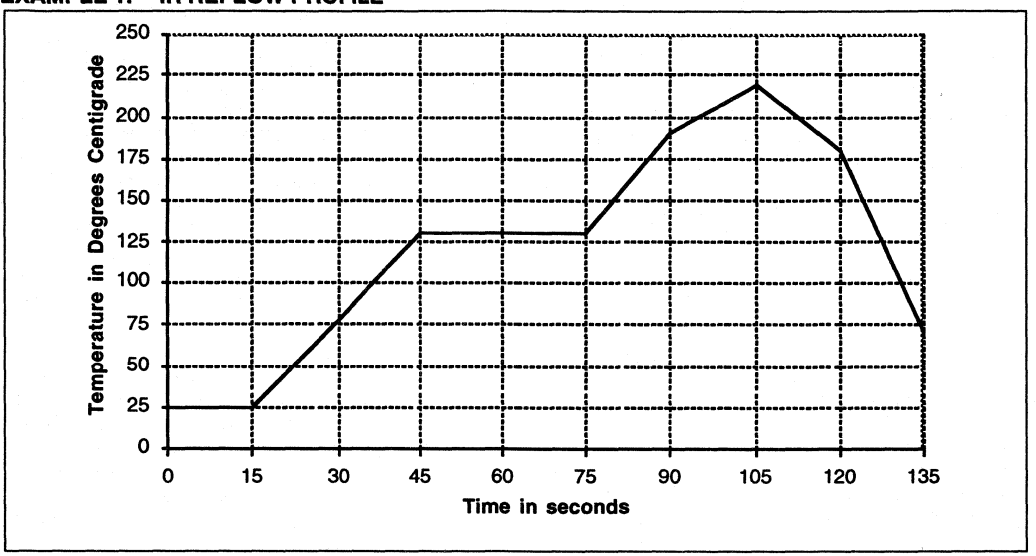
© 1997 Microchip Technology Inc.

TABLE 1: 32-LEAD PLCC PACKAGE PARTIAL FACTORIAL EXPERIMENT

Variables used in the analysis	Conditions							
M = Maximum Temperature (C)	220	235	250	265	280	295	310	7
R = Ramp Rate (C/second)	1.5	2	3	4				4
P = Number of Passes	1	2						2

56 Total Conditions

EXAMPLE 1: IR REFLOW PROFILE



1. (Continued)

No drypack state was employed and the parts were allowed to sit on the shelf in excess of 30 days prior to temperature exposure. The RH of the room was not recorded but is estimated to be approximately 60%. This condition was employed to insure that worst case conditions were evaluated.

Our experiments, to determine optimum soldering temperatures, indicated that a maximum temperature of 220°C should be observed during solder processing. The temperature ramp rate was found to be a less significant factor but should be kept in the range of 2 to 5°C per second. Very slow ramp rates, i.e. those of less than 2°C per second, show a slight decrease in performance. Ramp rates of 3°C showed the best performance and should be used whenever possible. Results from the experiment independently confirm the exposure recommendations set forth in IPC-SM-786[10]. A sample profile for IR reflow is illustrated by the graph in Example 1.

Prior to exposure to the IR temperature profiles all parts were electrically good and were examined by Scanning Acoustic Microscopy (SAM) to verify a defect free state and establish a baseline. After exposure to the IR profiles, all parts were again tested electrically and examined via SAM. Delamination can be easily detected by use of SAM. No package cracks were observed after temperature exposure. Only limited electrical failures occurred and all were at temperatures above 220°C.

In the experiments a thermocouple was interfaced to the exterior of a device and passed through an IR reflow oven. Both the maximum temperature and ramp rate were recorded in this manner. General trends for temperature sensitivity in relationship to maximum temperature and ramp rate are presented in the graphs shown in Example 2 and Example 3.

Pass/fail criteria for the experiments was primarily the extent of delamination, observed by Scanning Acoustic Microscopy, after exposure to two passes through an IR reflow oven. Top side die delamination and delamination in areas of the leadframe where the bonds are made are classed as unacceptable. Some delamination along non critical areas are classed as acceptable (i.e., no delamination extending to the exterior of the

© 1997 Microchip Technology Inc.

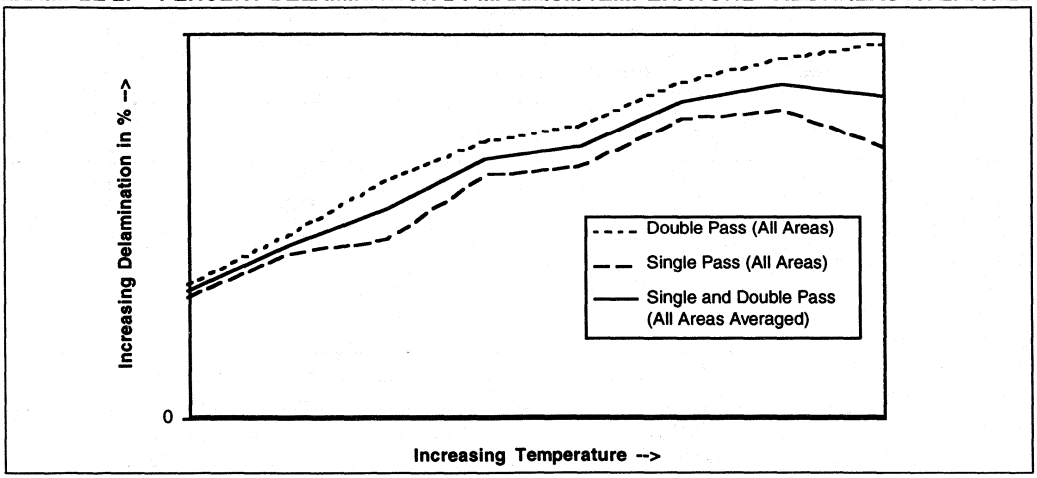
package or covering more than minor portions of the leadframe). Specific information on pass/fail criteria can be found in IPC - SM - 786.

Maximum temperature should not exceed 220°C.
Temperature ramp rate should be between 2 to 5°C per second.

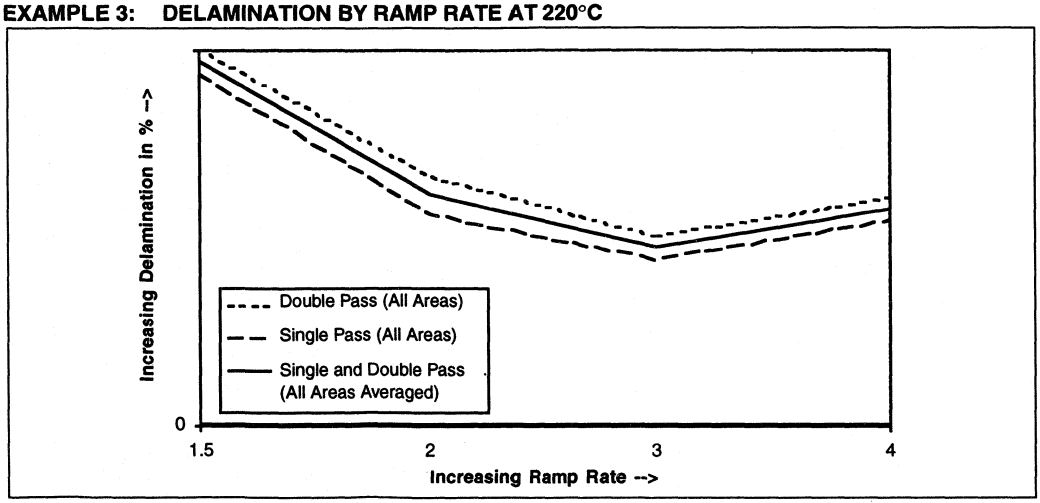
Use of the above temperature profile will increase the reliability of the device following board level assembly.

2. The second item that must be observed is the dry packing of SM devices. Dry packing is a standard product packaging procedure for surface mount devices. This practice guarantees to the customer that devices are shipped in a dry state. All devices are sealed inside plastic bags with a moisture indicator to monitor moisture when they are shipped. Customers should be aware of this practice and maintain the dry packed state until the time of use. Shelf life[11] for opened parts is a function of package style and ambient conditions. A small increase in failure rate will be experienced if the drypack condition is violated.

EXAMPLE 2: PERCENT DELAMINATION BY MAXIMUM TEMPERATURE - ALL AREAS AVERAGED



EXAMPLE 3: DELAMINATION BY RAMP RATE AT 220°C



8

© 1997 Microchip Technology Inc.

CONCLUSION

Significant thermal stressing can cause a diverse list of package or device failures. To minimize any damage caused by high temperature exposure, moisture content needs to be controlled in electronic devices.

Experiments conducted by Microchip validate the recommendation of IPC - SM - 786 to limit maximum temperature exposure during surface mount soldering to 220°C. A temperature ramp rate of 2°C/second to 5°C/second will also serve to safeguard the reliability of the device.

Microchip is committed to the principles of continuous improvement. Product reliability and customer satisfaction are a primary focus. For this reason, new packaging technology enhancements are continually evaluated to improve the performance of Microchip devices.

TABLE 2: PHYSICAL PROPERTIES OF COMMON MATERIALS USED IN PLASTIC PACKAGING OF SEMICONDUCTOR DEVICES

Material	Coefficient of Thermal Expansion in 10^6/°C (CTE)	Temp. of Glass Transition in °C (Tg)	Modulus of Elasticity in 10^6 psi (10^3 MPa.) (E)	Tensile Strength in 10^3 psi
Silicon (Si)	2.3-4.2		19-29	16-25
Alloy 42 Leadframe	4.0- 4.7		21	75
Alumina (Al_2O_3)	6.5-8.8		37	25-50
Gold Eutectic Die Attach	14.2		12.5	16
Copper Leadframe	16-18		17.3	60
Pb -- Sn Die Attach	29		1-2.5	1-4
Epoxy Mold Compounds (α_1 is for Temperatures below Tg, (α_2 is for temperatures above Tg)	7.5-28 α_1 20-22 ppm/°C α_2 60-65 ppm/°C[12]	145-170	1.8-3	10-20
Epoxy Die Attach	20-70		0.4	1-8
Polyimide Die Attach	35-60		0.6	1-8
Unfilled Epoxies	60-70			
Unfilled Silicones	300			

© 1997 Microchip Technology Inc.

ENDNOTES

1. Thomas M. Moore and Robert G. McKenna, *Characterization of Integrated Circuit Packaging Materials* (Stoneham: Butterworth - Heinemann, 1993), pp. 68.
2. A.S. Chen, et al., "A Study of the Interaction of Molding Compound and Die Attach Adhesive, with regards to Package Cracking", Proceedings of the 44th Electronic Components & Technology Conference. 1994, pp. 115-120.

   Michael Tencer, "Moisture Ingress into Nonhermetic Enclosures and Packages. A Quasi-Steady State Model for Diffusion and Attenuation of Ambient Humidity Variations", Proceedings of the 44th Electronic Components & Technology Conference. 1994, pp. 196-209.
3. Thomas M. Moore and Robert G. McKenna, *Characterization of Integrated Circuit Packaging Materials* (Stoneham: Butterworth - Heinemann, 1993), pp. 13.
4. George G. Harman, *Wirebonding In Microelectronics* (International Society for Hybrid Microelectronics, 1991), pp. 160.

   Justin C. Bolger, Short course notes. *Advanced Microelectronics Assembly Technology: Reliability & Yield Issues of Materials, Plastic Molding and Wire Bonding*, Univ. Of Ariz. February 8-11, 1994, pp. C2.
5. Samuel S. Kim, "Improving Plastic Package Reliability Through Enhanced Mold Compound Adhesion", Tutorial of the International Reliability Physics Symposium, 1992, pp. 2d.1-2d.17.
6. H. Ohsuga, et al., "Development of Molding Compounds Suited for Copper Leadframes", Proceedings of the 44th Electronic Components & Technology Conference. 1994, pp. 141-146.
7. Justin C. Bolger, Short course notes. *Advanced Microelectronics Assembly Technology: Reliability & Yield Issues of Materials, Plastic Molding and Wire Bonding*, Univ. Of Ariz. February 8-11, 1994, pp. C6.
8. T.M. Moore, S.J. Kelsall. "The Impact of Delamination on Stress-Induced and Contamination-Related Failure in Surface Mount IC's", *Proceedings of the International Reliability Physics Symposium.* 1992, pp. 169-176.

   T.R. Conrad and R. L. Shook. "Impact of Moisture/Reflow Induced Delaminations on Integrated Circuit Thermal Performance", Proceedings of the 44th Electronic Components & Technology Conference. 1994, pp. 527-531.
9. T.M. Moore, S.J. Kelsall. *Op. Cit.*, pp. 169-176.
10. IPC - SM - 786A is obtainable from the Institute for Interconnecting and Packaging of Electronic Circuits, 7830 N. Lincoln Ave., Lincolnwood, ILL. 60646.
11. IPC - SM - 786A, Table 4-2 (See endnote above).
12. Thomas M. Moore and Robert G. McKenna, *Characterization of Integrated Circuit Packaging Materials* (Stoneham: Butterworth - Heinemann, 1993), pp. 49.

An Excellent Reference Article --

L.T. Nguyen, "Reliability of Postmolded IC Packages", Transactions of the ASME, Journal of Electronic Packaging, December 1993, Vol. 115, pp. 346-355.

© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

MICROCHIP

AN603

Continuous Improvement

Author: *Randy Drwinga*
Microchip Technology Inc.

INTRODUCTION TO MICROCHIP'S CULTURE

The corporate culture at Microchip Technology Inc. is embodied in our *Guiding Values*. This culture has been key to our success in business because of its emphasis on customer satisfaction, quality, continuous improvement, empowerment and communication. The synergy of these values has created a very dynamic, very successful culture. They have impacted many aspects of Microchip, including the EEPROM Technology Team. Two of our *Guiding Values* in particular have steered this group.

Continuous Improvement is Essential

We utilize the concept of "Vital Few" to establish our priorities. We concentrate our resources on continuously improving the Vital Few while empowering each employee to make continuous improvements in their area of responsibility. We strive for constructive and honest self-criticism to identify improvement opportunities.

Products and Technology Are Our Foundation

We make ongoing investments and advancements in the design and development of our manufacturing process, device, circuit, system, and software technologies to provide timely, innovative, reliable, and cost effective products to support current and future market opportunities.

We recognized in the late 1980's that the industry as a whole did not have great endurance on EEPROMs. We were no exception. And this was at a 10,000 E/W cycle level, which is orders of magnitude less than where we are today. We also recognized that there were many uncertainties in the interaction of design, process and customers applications with respect to endurance performance and expectations. It was also clear that the customers were not educated enough on these interactions to be able to say clearly what they needed in a product. This was clearly an "improvement opportunity" that could allow us "to support current and future market opportunities."

THE EEPROM TECHNOLOGY TEAM

Microchip had been using the team approach to problem solving and continuous improvement activities such as new product introduction in the development groups and defect reduction in the manufacturing area. These had been in place for some time, and were beginning to work extremely well in bringing fast, well thought out and researched solutions to problems that crossed organizational boundaries.

A cross functional team was formed at Microchip that would be the technology management for an entire family of products. The team was comprised of engineers from design, process, test, product, quality, reliability, and yield enhancement, who reported in to different functional groups in the regular organizational structure. This team was charged with setting the technical direction for all EEPROMs at Microchip. This included both areas of development and sustaining, which were always in competition for the same resources.

The Microchip EEPROM Technology team went through the typical early phases of team development. The initial elation of having a great group of technical contributors wore off after a while. The competition for resources lead to some rough areas at the start, but this was overcome with the help of management support and direction. A balance developed, where the development side really understood the need to improve yields today, while the sustaining side understood the need for new products and innovations for the future. The emphasis on the Guiding Values and strong support from management for the team concept greatly helped the group through the early phases.

© 1997 Microchip Technology Inc.

THE PROCESS

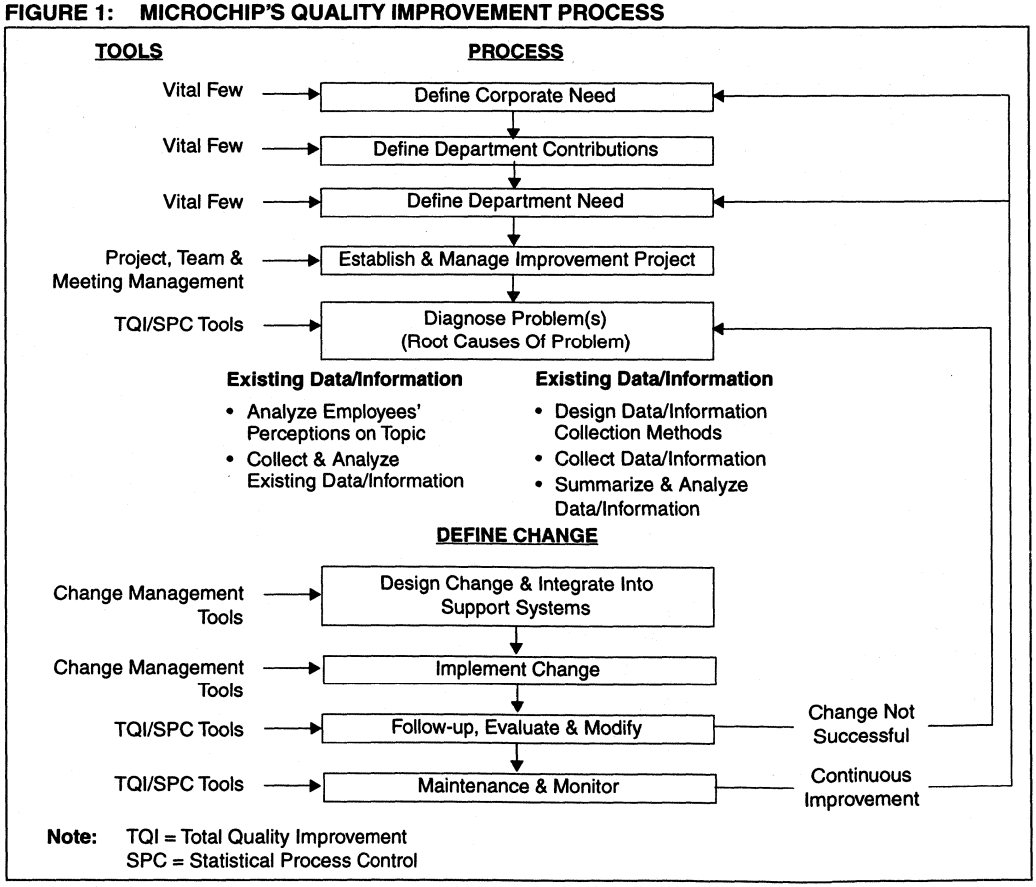
There is a team training program at Microchip that teaches high performance team leadership, and provides training on a set of tools for teams to use. These tools are aimed at improving the teams ability to identify problems, more importantly to identify the real root causes of the problems, to use the Pareto and other techniques to establish the "Vital Few", to brainstorm ideas on solutions, to investigate solutions prior to implementing, to execute implementations smoothly and accurately, to follow-up to ensure that the fix is as expected, and to re-examine the process to see what other improvements are needed. The key elements of this process are shown in the flowchart (Figure 1).

This process lead to improvements in every area of Microchip's EEPROM technology: design, process, product, test, quality, reliability, and yield. Most of these changes are confidential and proprietary, and several have been patented. These can only be discussed here in general terms.

The actual EEPROM cell design has been the subject of repeated studies. There have been improvements made to the original that have increased process tolerance, increased yield, and of course dramatically increased endurance performance. "Repeated studies" sounds as if we did not get it right the first time, but in fact is just an acknowledgment that continuous improvement is essential. We have the best design in the industry today, but the EEPROM technology team is still working on improving it.

The process also has gone through many iterations of improvement. During some of the early studies, it was determined that new equipment with better control was needed to ensure the reproducibility of the product. This equipment was purchased, installed, and carefully monitored based on the recommendation from the team. We saw the expected improvement as a results. There have also been ongoing studies to improve control, reproducibility, and defect reduction in many other areas of the process.

FIGURE 1: MICROCHIP'S QUALITY IMPROVEMENT PROCESS



© 1997 Microchip Technology Inc.

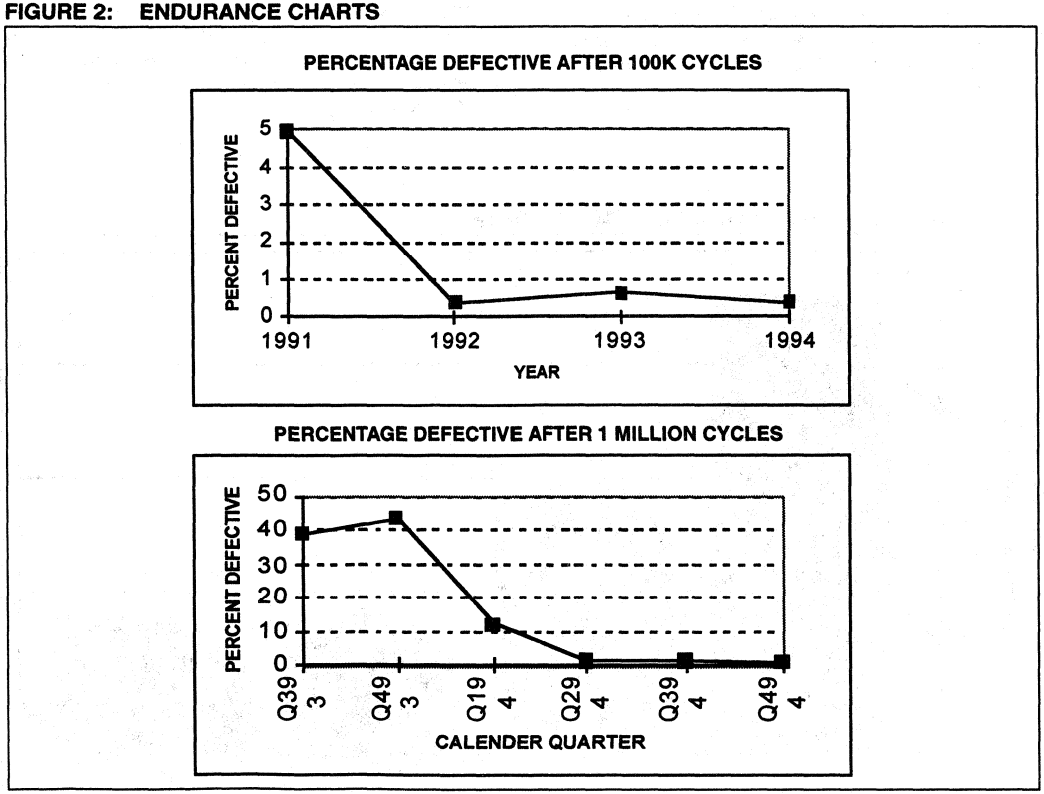
Improvements to the product itself have often shown up in the data sheet. These include reduction in current consumption, or improved timings as a result of improved design as verified by characterization on test equipment. We have greatly expanded the package variety for our Serial EEPROMs as a response to market needs. These activities have all been prioritized by the team with management input, evaluated, and implemented by the team.

The example of a new package introduction highlights the value of the team. A new package requires new test hardware, verification of the test program with the package and hardware, qualification of the package with the reliability group, assurance of acceptable yields based on the yield enhancement engineers involvement, and continued monitoring of the final products outgoing quality. Other companies often have these disciplines separated, with communication by memos back and forth. This is a cumbersome way to do business, and often details are overlooked, or the opportunity for synergy between groups to go the extra step is lost.

Testability is a vital concern for microelectronics. As the feature set grows, the ability to test them accurately can be lost. The introduction of the 64K-bit Serial EEPROM, with its security block and programmable endurance options, presented a concern for testability. By having even the initial design objective discussions in the team environment, with a test engineer, the designers were able to ensure that the new features could be tested thoroughly and cost effectively. This greatly improved our time to market, as well as the quality of the part that reached our customers.

The quality group has been a tremendous contribution to the success of the team. There were initial concerns on some members parts that this group would hold back progress by acting as "policeman" for the status quo. Quite the opposite, the quality group brought the statistical ability to measure the results of proposed changes with accuracy and confidence. So changes that really lead to improvement were actually implemented more quickly than they might have been without quality engineering input at the beginning. The team was also able to move on quickly from proposed changes that did not give measurable improvement as a result of having these measurement tools.

FIGURE 2: ENDURANCE CHARTS



8

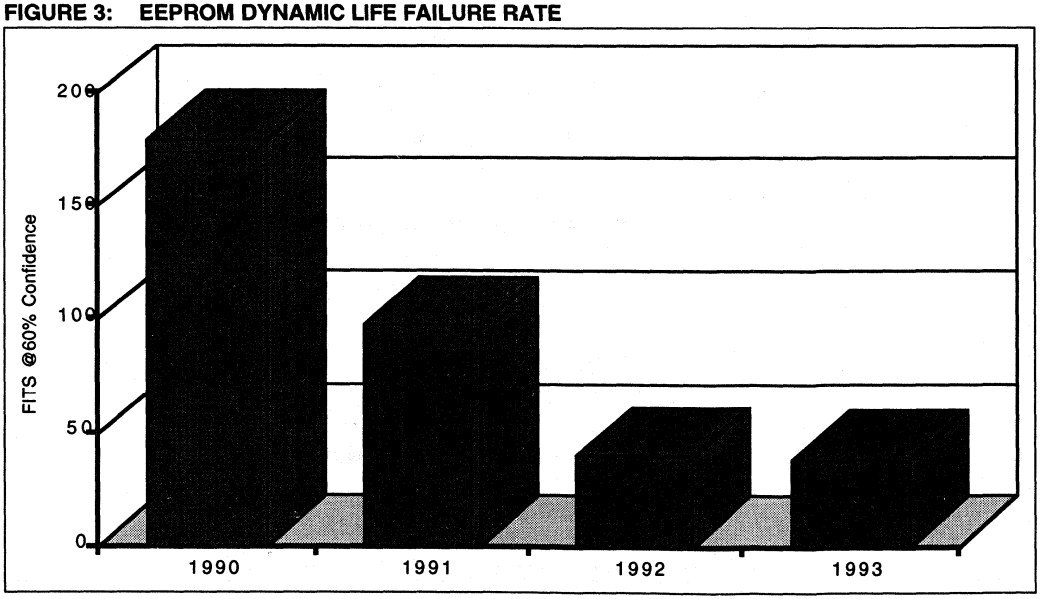
The quality engineering group is also more directly exposed to customer needs than some of the other groups. They heard first hand what some of the areas for improvement were from the field. These inputs were brought directly into the team environment, where the breadth of engineering expertise we had was able to apply Microchip's Quality Improvement Process to them.

The dramatic improvement to the endurance performance of Microchip's Serial EEPROMs was a direct result of the reliability group's involvement with the team. This group was able to establish standard methods for testing and measuring endurance on products. This was the first step in determining that this was another area for continuous improvement. Inputs from the business unit were that this was going to develop into a critical point of competition in the industry. We found out later that our standards were significantly higher than most competitors. The reliability group set up competitive analysis and bench marking between vendors that put us on a level playing field. We found that Microchip's endurance was much better than we thought! Even so, the Microchip EEPROM Technology team drove programs for improving endurance that gave impressive results (Figure 2). And the elimination of the root causes of endurance problems gave equally impressive gains in other areas of reliability. This is shown is the reduced failure rates for both dynamic life testing and retention bake (Figure 3 and Figure 4).

The group also kept one eye on the bottom line through the years. Yields have always been a concern, since continuous improvement in that area is also vital to the business. The yield enhancement engineers, like the other groups, brought tools with which to measure and identify problems. As with the others, they were dependent on the rest of the team to help find and implement solutions. They also acted as a conduit to the front end teams that were implementing defect density reduction and process control improvements in the manufacturing process across product families. This coordination between the teams lead to a whole new level of synergy that allowed other products to benefit from improvements from the EEPROM team, and the EEPROM product line to piggyback on improvements aimed primarily at other areas. These other teams were focused on plant wide areas of continuous improvement: oxide integrity, the poly-silicon modules, the metal modules, defect reduction, wafer handling techniques, etc. The result of this cross functional synergy is shown is the yield trend (Figure 5).

As we grew and learned together, each member developed increasing respect and trust for the others. This was critical to the team's development. Not all of the proposed changes were successful. In the stages of implementing, follow-up, and maintenance, some proposals did not hold up. Either they did not really fix a root cause, or the implementation caused other problems to surface. Only by having established respect and trust was the team able to raise and address these issues quickly.

FIGURE 3: EEPROM DYNAMIC LIFE FAILURE RATE



© 1997 Microchip Technology Inc.

THE RESULTS

The improvements to both areas that this team has made are spectacular. At the time of formation, the industry standard was 10,000 E/W cycles of endurance, and 4K bits of memory. The Microchip lead the world when the efforts of the team lead to introducing:

- The first 10,000,000 E/W cycle guarantee
- The very first 64K bit Serial EEPROM
- The first to put a 64K bit Serial EEPROM in an 8-pin SOIC package
- The first family of Serial EEPROMs to operate at 1.8V
- The first VESA busing compatible Serial EEPROMs

These improvements are orders of magnitude increases in critical areas of market need. They establish Microchip Technology Inc. as the technical leader in EEPROM technology. This is what the EEPROM Technology team set out to do.

The team's efforts continue today to make sure Microchip remains a technical leader. We have the best products in the market today. But that will not be good enough tomorrow. Continuous improvement is essential.

FIGURE 4: EEPROM RETENTION BAKE FAILURE RATE

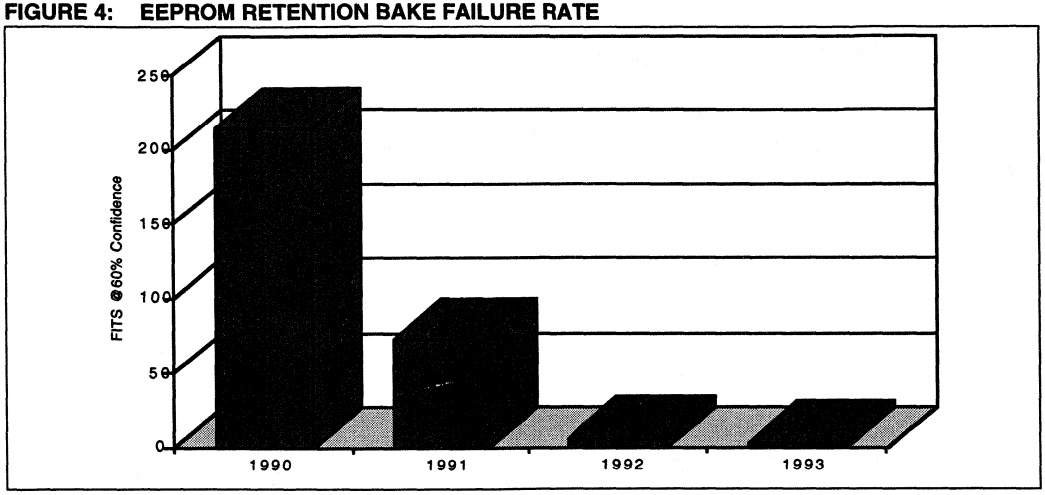
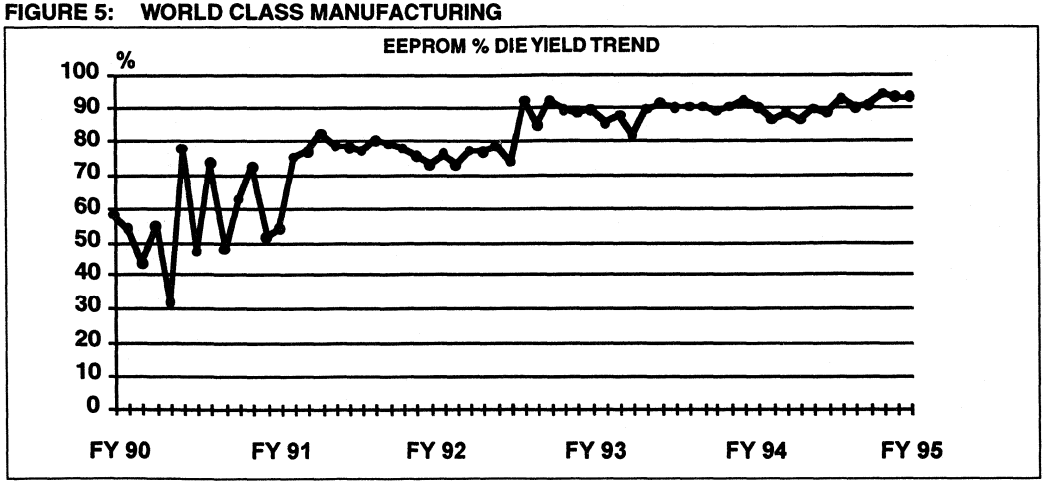


FIGURE 5: WORLD CLASS MANUFACTURING



8

© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

SECTION 9
TECHNICAL BRIEFS

Secure Learning RKE Systems Using KEELOQ® Encoders - TB001 9-1
An Introduction to KEELOQ® Code Hopping - TB003 9-7
Automatic Calibration of the WDT Time-out Period - TB004 9-17
Using External RAM with PIC17CXX Devices - TB005 9-23
Transformerless Power Supply - TB008 9-31
Adding a Simple 4-channel 8-bit A/D to a PIC17C4X - TB010 9-33
Using SRAM With A PIC16CXX - TB011 9-45

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

MICROCHIP

TB001

Secure Learning RKE Systems Using KEELOQ® Encoders

Author: Chris R. Burger
Microchip Technology Inc.

INTRODUCTION

Learning capability in Remote Keyless Entry (RKE) and remote-controlled security systems is regarded as essential by most manufacturers. The logistical problems associated with the supply of replacement and additional transmitters for personalized decoders quickly become overwhelming if any dealer intervention is required.

In the case of a learning system, the user can purchase a preprogrammed transmitter off the shelf and then add that transmitter to the decoder system without assistance. Dealer intervention is completely unnecessary, and only one type of transmitter needs to be stocked for a particular product line. Each transmitter is pre-programmed with a unique serial number and key.

However, learning systems need to be properly managed to ensure that they maintain an adequate level of security. A badly implemented learning system could provide an outsider with access to the system. On the other hand, a well-designed learning system should not reduce the security level of the basic code-hopping system at all.

SINGLE-ALGORITHM SYSTEMS

Code hopping systems often use a single encoding and decoding algorithm for all transmitters in a particular product line. Most of the systems on the market fall into this category.

Learning is really simple—the decoder simply decodes the incoming transmission during learning and stores the resultant parameters for later use. For their security, these systems rely on the assumption that the algorithm will remain secret. In this era of Internet and instant worldwide communications, the probability that a widely-used algorithm will permanently remain secret is low and the assumption naive.

9

KeeLoq is a registered trademark of Microchip Technology Inc.

© 1997 Microchip Technology Inc.

THE KEELOQ KEY-BASED SYSTEM

The KEELOQ system uses a separate 64-bit key for each transmitter. Such a key is simply a very large random number, unique to that transmitter. Effectively, this arrangement provides a unique encoding and decoding algorithm for each transmitter. An outsider that does not know the key, cannot decode the variable code portion of a transmission and consequently cannot determine the identity parameters of the originating transmitter.

This uniqueness of each transmitter's encoding algorithm complicates learning. The key cannot be determined from the variable code transmission, and no information can be derived from the transmission without the key.

Obviously, some other piece of information must be transmitted during learning to enable the decoder to calculate the correct key. Two approaches are suggested. Each approach has pros and cons, which will be the subject of a comparative discussion in a later section.

KEELOQ Normal (Serial Number-derived) System

Each KEELOQ transmitter contains a unique serial number, programmed into the transmitter on the production line. This serial number is transmitted as part of the fixed code portion of every transmission.

When a transmission has been received, decoders typically use this serial number to determine the identity of the originating transmitter. The serial number is compared with those stored in memory. If a match is found, the decoder knows the identity of the transmitter, and therefore also knows which key and counter to use to process that transmission.

Each transmitter is also programmed with a key, calculated from the serial number using a secret learning algorithm. The relationship between the serial number and the key is very complex. This ensures that the relationship is not evident to outsiders.

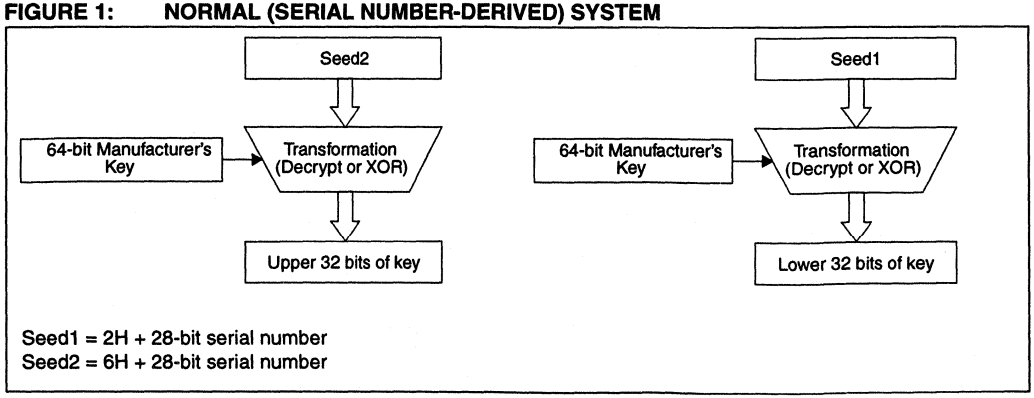
During learning, the decoder calculates the key for that transmitter, using the same secret learning algorithm used for programming the transmitter. Once the key has been determined, the decoder can decode the transmission and store the parameters associated with that transmitter, including the key.

This learning procedure offers simplicity and high security—provided that the learning algorithm remains secret. To reduce the possibility of the learning algorithm being jeopardized, the KEELOQ system uses a learning algorithm that is only implemented in custom ICs. The first decoder to use this algorithm is slated to become available during 1996. As an interim solution, customers with a requirement for unusually robust learning security can enquire about coprocessor-based solutions.

In addition, the system relies on a manufacturer's key to determine the learning relationship. The manufacturer's key is protected by a smart card-based system and is stored in EEPROM inside the custom IC. Even if the learning algorithm itself becomes known, each manufacturer has a second line of defence in the manufacturer's key. Should a single manufacturer allow their key to become public knowledge, other manufacturers are not endangered.

One final comment—the envelope encryption capability on some of the KEELOQ encoders does not materially alter the nature of the learning algorithm. All devices in a particular product line share a single envelope encryption key, and any decoder in that product line can readily decode an incoming serial number. Once the serial number has been determined, the learning algorithm proceeds exactly as detailed.

FIGURE 1: NORMAL (SERIAL NUMBER-DERIVED) SYSTEM



© 1997 Microchip Technology Inc.

KEELOQ Secure (Seed-derived) System

The ultimate in secure learning is a system where no reliance is placed on the secrecy of any of the algorithms, or a single manufacturer's key.

The KEELOQ code hopping system was designed under this assumption. Even if an outsider has the code hopping algorithm, a particular transmitter's transmissions are still incomprehensible if that transmitter's secret key is not known.

Determining the key by analyzing a number of transmissions is also not feasible. In 1995, it was estimated that an attacker with access to the algorithm requires a custom-designed $1,000,000 computer (designed exclusively to analyze KEELOQ transmitters) and 37 days of computer time per transmitter to find the secret key. Also, if a particular transmitter is jeopardized, no harm has been done to the security of other transmitters, even from the same product line.

To extend the security advantages of open algorithms to learning systems, the KEELOQ developers have applied for patents covering a novel learning technique. The learning technique does not rely on the secrecy of the learning algorithm at all.

On the production line, each transmitter is programmed with a serial number, a learning seed, and a key. There should not be any deterministic or mathematical relationship between the serial number and the key. Instead, a fixed (but complex) relationship exists between the learning seed and the key. The learning seed is only transmitted during learning. A special action is required from the user to activate transmission of the learning seed.

The learning seed is never transmitted during normal operation.

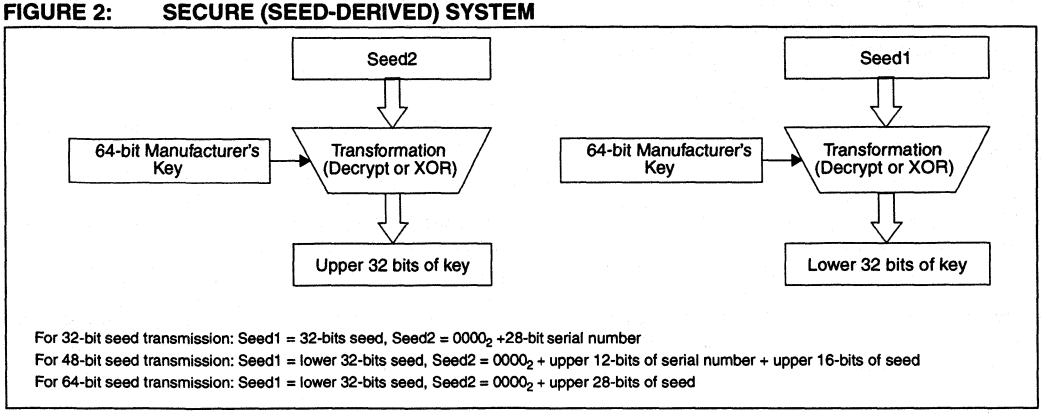
The HCS300 encoder can transmit a learning seed when all the function inputs are activated simultaneously. The 32-bit variable code is then replaced by a 32-bit learning seed, retrieved from the encoder's EEPROM memory. The decoder can derive the key from the learning seed alone, or from both the learning seed and the serial number.

Seed transmission in the HCS360 can be activated in two ways, details of which can be found in the following section and in the device specifications. During seed transmission, the HCS360 replaces both the 32-bit variable code and 16 bits of the serial number with fixed information retrieved from EEPROM, in essence transmitting a 48-bit seed. Also, additional protection against attack is provided. If desired, the transmitter can be configured to completely lose its ability to transmit the learning seed once the learning process has been completed. The mechanism works by permanently disabling seed transmission capability when the synchronization counter reaches 128. The user does not need to take any conscious action, as seed transmission is automatically inhibited after a few normal code hopping transmissions and cannot be activated again unless the encoder is reconfigured in total.

Because the learning seed and the key are both stored in read-protected EEPROM, there is no way to obtain the learning seed or the key from the transmitter, once the seed transmission capability has been inhibited.

A major advantage of seed based learning systems is that the security is not reliant upon a single key (or algorithm) that must be present in every decoder. However, a secret manufacturer's key that determines the relationship between the seed and the key still ensures protection against situations where access to the transmitter is possible (i.e. servicing, valet, etc.) and against the manufacturing of pirate transmitters.

FIGURE 2: SECURE (SEED-DERIVED) SYSTEM



For 32-bit seed transmission: Seed1 = 32-bits seed, Seed2 = 0000_2 +28-bit serial number
For 48-bit seed transmission: Seed1 = lower 32-bits seed, Seed2 = 0000_2 + upper 12-bits of serial number + upper 16-bits of seed
For 64-bit seed transmission: Seed1 = lower 32-bits seed, Seed2 = 0000_2 + upper 28-bits of seed

9

© 1997 Microchip Technology Inc.

USING LEARNING SEED TRANSMISSION (HCS300, HCS301, HCS200, HCS360, AND HCS361)

HCS300/301

The HCS300 transmits a fixed code (stored in EEPROM) when all four control inputs are activated (i.e. $S_3S_2S_1S_0 = 1111$).

HCS200

The HCS200 transmits a fixed code (stored in EEPROM) when all three control inputs are activated (i.e. $S_2S_1S_0 = 1$).

HCS360/361

In the HCS360, the seed transmission capability is optional. If this option is selected during programming, transmission can be initiated in two ways: either $S_3S_2S_1S_0 = 0011$ and delayed mode is active (i.e. after about 3 seconds of variable code transmission), or $S_3S_2S_1S_0 = 1000$.

The fixed code capability can be permanent or temporary, depending on the setting of another EEPROM configuration bit. If the temporary mode has been selected, fixed code capability is disabled when either of the hopping code counters reaches a value of 128. The user can transmit the learning seed as many times as required to complete learning, and then originate up to 128 code hopping transmissions. The HCS360 will then protect the learning seed against readback and transmission for the remainder of its lifetime. If a number of transmissions less than 128 is required, the initial counter value can be increased accordingly.

Transmission Format

Normal HCSxxx encoder transmissions consist of a 32-bit hopping code, a 28-bit serial number, a 4-bit function code, and a flag field. These bits include a low voltage warning flag, a transmission repetition flag, and CRC bits for error checking. The flag field differs for the two encoders, and is not germane to this discussion. More information appears in the specification documents for each of the encoders.

H_0	Hopping	H_{31}	N_0	Serial	N_{27}	$S_2S_1S_0S_3$	Flags

The HCS200's and HCS300's seed transmission mode is identical, except that the 32 bit variable code is replaced by a 32-bit seed value, retrieved from EEPROM.

K_0	Seed	K_{31}	N_0	Serial	N_{27}	$S_2S_1S_0S_3$	Flags

In the HCS360's seed transmission mode, the fixed code is composed of a 48-bit learning seed, bits 16 to 27 of the serial number (the first 16 bits are replaced by seed bits), the 4 function bits, and the flag field.

K_0	Learning seed	K_{47}	N_{16}	Serial	N_{27}	$S_2S_1S_0S_3$	Flags

If compatibility between HCS300/301 and HCS360/361 transmitters is required, the HCS360 can simply be programmed so that the upper portion of the seed (bits K_{32} to K_{47}) corresponds to the lower portion of the serial number (bits N_0 to N_{15}). The resulting transmissions are then identical, except for possible differences in the flag field.

© 1997 Microchip Technology Inc.

DECIDING ON A LEARNING SOLUTION

Factors to be Considered

Any security system is a compromise between convenience to the user, cost and security. The KEELOQ system has made very high security available at low prices, all but eliminating cost as a consideration. The designer must therefore decide on the relative importance of security and user-friendliness in the system.

If security is of paramount importance, a seed-based system with automatic seed inhibition is preferred. However, this system has the disadvantage that the user must place both the transmitter and the receiver in learning mode, and that the transmitter can only be learned by a decoder once during its lifetime.

If user-friendliness is more important, a seed-based system without seed inhibition or a serial number-based system can be used. In the case of a serial number-based system, the user does not have to memories any special button combinations for use exclusively during learning. In the case of a seed-based system, the user needs to know the special button combination, but the transmitter retains its learning capability indefinitely.

TABLE 1: PROS AND CONS OF THE DIFFERENT LEARNING SYSTEMS

Learning Mode	How Used	Advantages	Disadvantages
Serial Number-based Learning	During learning, the key is derived from a serial number, included as part of every transmission from the transmitter.	The user does not need to activate a special encoder mode to conduct learning. Normal transmissions are used during learning, and the key is derived from the included serial number information. Also, a transmitter can be re-learned at any time if required.	The security of the system is dependent on the secrecy of the learning algorithm and/or manufacturer's key. This disadvantage can be overcome by using a learning algorithm implemented in a custom IC. However, for the largest OEM product lines, syndicates may still find it worth their while to reverse-engineer the custom IC.
Seed-based Learning With Seed Inhibition	During learning, a special learning seed is transmitted. The decoder derives the key from this learning seed. During normal operation, the transmitter loses its ability to transmit the learning seed. The seed is also stored in read-protected EEPROM, fully protected against outside access.	The security of the system is independent of the secrecy of the learning algorithm. The learning algorithm can thus be implemented on any platform, including generic microprocessors, without fear of jeopardizing the security of the system.	The user must operate a special button or combination of buttons on the transmitter to transmit the learning seed. Also, the transmitter cannot be re-learned once seed transmission has been disabled.
Seed-based Learning Without Seed Inhibition	During learning, a special learning seed is transmitted. The decoder derives the key from this learning seed. During normal operation, the transmitter does not transmit the learning seed. The system is therefore not susceptible to outside attack, even from someone that knows the learning algorithm and manufacturer's key. However, the transmitter permanently retains its ability to transmit the learning seed, and can be re-learned at any time.	The security of the system is independent of the secrecy of the learning algorithm, as the learning seed is not transmitted during normal operation. The learning algorithm can be implemented on any platform, including generic microprocessors, without fear of jeopardizing the security of the system. The transmitter can be re-learned at any time, as required.	The user must operate a special button or combination of buttons on the transmitter to transmit the learning seed. Also, there is some risk of the learning seed being revealed, as an outsider with temporary access to the transmitter can cause the transmitter to transmit the learning seed.

© 1997 Microchip Technology Inc.

IMPLEMENTATION ISSUES

This section presents hardware and software issues surrounding various implementations and should be read as a guide to implementation once a solution has been chosen.

Serial Number-based Systems

Proceed to implement a decoder as indicated in the relevant KEELOQ documents. Pay attention to the platform being used. ROM-based microprocessors should only be used as a last resort. If possible, use a KEELOQ decoder or coprocessor to ensure that the learning algorithm remains secret.

Three stages can be identified in the learning process. These three stages involve two different transmissions. The user presses the button, causing a normal code hopping transmission from the transmitter. During the first stage, the serial number is stored in EEPROM, and the corresponding key is calculated. During the second stage, the decoder decodes the incoming transmission using that key and stores the decoded parameters (function, integrity testing information and synchronization counter) in EEPROM. Some form of user feedback is then provided, prompting the user to press the transmitter button again. The third stage consists of decoding the resulting transmission, comparing the integrity testing information to the stored version, and ensuring that the two counter values are successive.

If code space is at a premium, or the simplicity of the user interface is paramount, the second transmission (and hence the third stage) can be omitted. Some integrity is sacrificed, as the second transmission is used to ensure that the transmitter's key has been correctly calculated and that the transmitter actually belongs to the same product line as the decoder. If the second transmission is forfeited, the system designer should ensure that the integrity testing information bits are subject to some convention, failing which any incoming transmission would be accepted as valid during learning. A possible programming convention is to use the lower 12 bits of the serial number as integrity testing information.

Learning Seed-based Systems

a) Decide on the user interface during learning. Would it make more sense to press a separate secret button, which normally requires disassembly of the transmitter, or to press a combination of two buttons?

   For two button transmitters based on the HCS360, no additional hardware is required to implement a secure learning system. If the two transmitter buttons are pressed together, the transmitter transmits a normal hopping code, and then reverts to fixed code mode after approximately 3 seconds. The decoder can determine the 28-bit serial number from the initial transmission and then calculate the key from the learning seed when the transmitter reverts to fixed code encoder mode.

   During a further transmission, the decoder can decode the incoming transmission to determine and store the counters and integrity testing information. The designer may elect to include a third transmission to verify the correctness of the decoding operation. Similar considerations to those mentioned in Section apply.

   After the predetermined number (up to 128) of code hopping transmissions has been made, the transmitter loses the ability to transmit the learning seed. From this point, pressing the two buttons together for more than 3 seconds results in a delayed function transmission with a function code of 0011.

   For other HCS360-based transmitters, activation of S3 results in transmission of the learning seed. S3 may be activated by installing a temporary link on the board, or even by a separate push button.

   After the predetermined number (up to 128) of code hopping transmissions has been made, the transmitter loses the ability to transmit the learning seed. From this point, activating S3 results in a normal hopping code transmission with a function code of 1000. For multi-button transmitters, this option opens up the possibility of using a normal button (i.e. fully accessible from outside) for learning, as the button regains full functionality after the fixed code transmission mode is disabled.

   For HCS300/301-based systems, all control inputs (S_0 to S3) must be high to activate seed transmission. The designer should include special provisions for forcing all these inputs high, especially in the case of a transmitter with less than four buttons.

b) Modify the decoder algorithm to calculate the key from the learning seed rather than from the serial number.

c) Choose the number of hopping code transmissions allowed before the encoder loses the ability to transmit the learning seed.

   Remember that the transmitter cannot be re-learned once the fixed code transmission has been disabled. If permanent re-learning capability is required, fixed code transmission should be left active permanently, or a serial number-based learning technique should be implemented. In most consumer products, the security level offered by the normal learning technique is perfectly adequate and sacrificing the convenience of re-learning is not justified.

© 1997 Microchip Technology Inc.

MICROCHIP

TB003

An Introduction to KEELOQ® Code Hopping

Author: Kobus Marneweck
Microchip Technology Inc.

INTRODUCTION

Remote Control Systems

Remote control via RF or IR is popular for many applications, including vehicle alarms and automatic garage doors. Conventional remote control systems are based on unidirectional transmission and have limited security. More sophisticated devices based on bi-directional transmission are also available but, because of their high cost and certain practical disadvantages, they are not widely used in commercial remote control devices.

The popular unidirectional transmission systems currently have two very important security shortcomings: the codes they transmit are usually fixed and the number of possible code combinations is relatively small. Either of these shortcomings can lead to unauthorized access.

Code Scanning

The limited number of possible combinations available in most remote control systems makes it possible to transmit all possible combinations in a relatively short time. A hand held microprocessor-based system for this purpose (called a code scanner) can easily be constructed.

In systems using eight DIP switches (256 combinations), this scanning process can typically be accomplished in less than 32 seconds (when trying eight combinations per second). Even in systems using 16-bit keys (yielding roughly 65,000 combinations), only 2.25 hours would be required to try all possible combinations. It should also be noted that the scanner may gain access in far less than this maximum time—the average time would in fact be half of the total time.

Scanning is counteracted by increasing the number of possible code combinations. A 66-bit code will yield 7.3×10^{19} combination and will take 2.3×10^{11} years to scan.

Code Grabbing

A far easier way of gaining unauthorized access to a security system is freely available—such a unit is being advertised as a tool for the "legal repossession of vehicles." To understand its operation, it is useful to know something about remote controls.

A remote control transmitter of the type normally used in vehicle security systems, is nothing but a small radio transmitter that transmits a code number on a certain frequency. This code number is normally generated by an integrated circuit encoder. The transmit frequency is normally fixed by legislation within a particular country, enabling anybody to build a simple receiver that can receive signals from all such transmitters.

It is a simple matter to build a circuit to record such transmissions captured by the receiver. Such a device is known as a code or key grabber. A would-be vehicle thief would typically lurk in a parking lot, waiting until a vehicle owner arms his alarm with a remote control. The key grabber would capture the transmitted code, enabling the thief to retransmit this code as soon as the owner leaves the parking lot. Typically, this would leave the alarm and/or immobilizer disabled and even the central locking unlocked.

The Solution

It is apparent that secure remote control systems can only be implemented if two conditions are met. The KEELOQ® code hopping system meets both these conditions with ease.

- A large number of possible combinations must be available.

  A 66-bit transmission code is used to make scanning impossible. The 32-bit encrypted portion provides for more than 4 billion code combinations. A complete scan would take 17 years! If the 34-bit fixed portion is taken into account, the time required for a complete scan jumps to 5,600 billion years!
- The system may never respond twice to the same transmitted code.

  The random code algorithm will never respond to the same code twice over several lifetimes of a typical system.

Every time a remote control button is pushed, the system will transmit a different code. These codes appear random to an outsider – there is no apparent relationship between any code and the previous or next code.

Once the system has responded to a valid code, about 65,000 valid codes will have to be received before the same code will be used again. If the remote control is used eight times daily, 22 years will pass before the system responds to the same code again - once! Therefore, a retransmitted code (like when a code grabber is used) will never activate the system.

KEELOQ is a registered trademark of Microchip Technology Inc.

© 1997 Microchip Technology Inc.

The KEELOQ algorithm also features sophisticated synchronization techniques. The system will continue to function even if the transmitter is activated repeatedly while not in range of the receiver (as would happen if a child played with the remote control). If a button is pressed out of range more than 16 times, synchronization will be lost. However, two successive transmissions in range will restore synchronization. When no response occurs to a transmitter operation, the user's natural reaction is to press the button a second time. Synchronization will be restored when he does. Operation is totally transparent — the user may not even become aware that synchronization has been lost and restored.

These obvious security benefits are attainable at a cost comparable to existing fixed code systems. The revolutionary nature of the KEELOQ algorithm, full custom IC implementation, and reduced external component requirements help to make this level of cost effectiveness possible.

The KEELOQ Algorithm

KEELOQ is a proprietary block cipher based on a block length of 32 bits and a key length of 64 bits. The algorithm is characterized by a very economical hardware implementation, while retaining a level of security comparable to Data Encryption Standards (DES). This level of security makes it eminently suitable for code communication applications such as code hopping antitheft or access control devices.

Information regarding transmitter identity and synchronization is encoded so as to render it unintelligible to an outsider. For decoding, it is necessary to have the same 64-bit key originally used for encoding. Therefore, even though the decoder (which has the key) can identify the transmitter unambiguously, an outsider (who does not have access to the key and/or the algorithm) can glean no information at all from the transmissions.

As it is impossible to insert information into the system from outside, strategies used to attack FEAL and other DES-like ciphers are not usable against this system.

The KEELOQ algorithm is designed to make it impossible for a potential assailant to predict the next code that will be transmitted by a valid transmitter. Even if the assailant makes a reasonable guess regarding the way in which transmitted information changes with each transmission, the algorithm obscures this information sufficiently that the next code can not be anticipated. In particular, even if the transmitted information (before encoding) differs only in one bit from the information in the previous transmission, the next transmission will be totally different.

Checks exist that can be used to verify the security characteristics of an encoding algorithm and, in this instance, to determine whether the next transmitted code is predictable to any degree. The Avalanche Effect and a subset thereof, the Strict Avalanche Criterion, have been tested on the KEELOQ algorithm. The results give a good indication of the security offered by the system.

- Avalanche Effect (AE)

  A block cipher satisfies the AE if changing one bit of the information causes, on average, half of the bits in the transmission to change. In the KEELOQ algorithm, this implies that changing one bit in the function and/or synchronization information will cause an average of 16 of the 32 bits in the transmitted code to change.

- Strict Avalanche Criterion (SAC)

  The SAC requires that, if one bit of the encoded information is changed, each bit in the output must have a chance of 0.5 of changing as well. Consequently, the probability of guessing any one bit correctly is 0.5, and the probability of guessing an entire 32-bit string correctly is one in about 4,300,000,000!

  The tests involved using a random 64-bit key and a Gray counter (starting at zero) as input to the algorithm. In each case, the output was compared with a reference (SAC) or with the previous code (AE). In both cases, the results were as expected: For the AE, the average number of bits changed was 16.0 (50%), with a standard deviation of 2.83 (8.8%). For the SAC, each individual bit changed an average of 50% of the time, with a standard deviation of 8.8%.

© 1997 Microchip Technology Inc.

DEFINITIONS USED IN CODE HOPPING

Transmitter Serial Number	Each transmitter is programmed with a unique 28- or 32-bit serial number at the time of transmitter manufacture. This ensures that each transmitter will be unique within a system.
Secret Key	This 64-bit secret key is generated by a key generation function from the 28- or 32-bit serial number or the 32- or 48-bit seed and 64-bit manufacturer's key as inputs. A manufacturer's key can be used to control the key generation function. The secret key is not readable and is never transmitted.
Seed	The seed is a 32- or 48-bit value that is programmed into the encoder. This can be programmed to be the same as one half of the key or be used in key generation. It is only transmitted when a specific button combination is activated and can be disabled once learning is completed.
Key Generation	The key generation function is used to generate a unique key for each transmitter from the serial number or seed value. It is preferable that the function is nonlinear. This function is performed by the programming station to generate keys to be programmed into transmitters.
Manufacturer's Key	The manufacturer's key is necessary in the receiver if a key generation function with a manufacturer's was used to generate the secret keys. The manufacturer's key must be programmed into the receiver during manufacture.
Normal Learn (Serial Number Derived)	The receiver uses the same information that is transmitted during normal operation to derive the transmitter's secret key, decrypt the discrimination value and the synchronization counter. All the transmitter information is then stored.
Secure learn (Seed Derived)	The transmitter is activated through a special button combination to transmit a stored 32- or 48-bit value (seed) that can be used for key generation or be part of the key. Transmission of the random seed can be disabled after learning is completed.
Discrimination value	The discrimination value is a 12-bit fixed portion of the encrypted word. It is used as a post decryption check.
Synchronization counter	A16-bit counter that is incremented on every activation of the encoder. It is stored on the receiver controller and compared to determine if a transmission is a previously received transmission, in which case it is ignored or falls within a forward window in which case it is accepted.

© 1997 Microchip Technology Inc.

FEATURES

A full perspective of the capabilities of the KEELOQ system would involve describing the unique features of each device separately. However, the entire product range has certain characteristics that may or may not be applicable to a specific member of the product range. The most important characteristics are listed to enable the prospective user to get a global view of the KEELOQ system. The specifications for a particular encoder or decoder should be consulted to determine the applicability of these features to that device.

Encoders

Security

- Programmable 28/32-bit serial number
- Programmable 64-bit encryption key
- Each transmission is unique
- 66/67 bit transmission length
- 32-bit hopping code
- 34/35-bit fixed code (28/32-bit serial number, 4/0-bit function, VLOW, Rpt/2-bit CRC)
- Encryption key is read protected

Operating

- One (3.0V) or two (6.0V) Lithium battery operation
- One (9.0V or 12.0V) Alkaline battery operation
- 3 or 4 button inputs - 7 or 15 functions
- Selectable baud rate
- Automatic power down after transmission
- Battery low signal transmitted
- Nonvolatile synchronization data
- IR modulation capability

Other

- On-chip EEPROM
- On-chip oscillator and timing components
- On-chip reset circuit
- Button inputs have internal pull-down resistors
- Current limiting on LED output

Decoders

Security

- KEELOQ code hopping technology
- Secure storage of manufacturers key
- Secure storage of transmitters keys
- Normal learning mode and Secure learning mode
- Four or more transmitters can be learned

Operating

- 2.0V to 6.0V operation
- 4 MHz RC oscillator
- Learning indication
- Auto baud rate detection

Decoder

- Single chip decoder available
- On-chip EEPROM
- Four binary function outputs - 15 functions
- One or two wire serial interface
- Battery low indication

TYPICAL APPLICATIONS

- Automotive remote entry systems
- Automotive alarm systems
- Automotive immobilizers
- Gate and garage openers
- Electronic door locks
- Identity tokens
- Burglar alarm systems
- Remote control toys/fixed code applications

ENCODER FUNCTIONS

General Description

The KEELOQ code hopping remote control encoders are intended for secure remote control systems. They are suitable for use in remote control applications using infrared (IR), microwave or other radio frequency (RF) transmitters.

The encoders have several basic modes of operation. Some of these modes may be combined in a particular device, and the means of selecting the mode may differ, but the principle of operation remains the same.

Unidirectional Authentication

Unidirectional authentication is used where it is impractical or cost prohibitive to use a bi-directional link such as remote keyless entry. In this mode, an encoder functions as a code generator for a transmitter. A suitable transmission medium (RF, ultrasonic, microwave, infrared, or magnetic coupling) must be provided in addition to the KEELOQ encoder to implement a complete link. The encoder is activated by a push button or by a magnetic field in the case of a transponder.

Bi-directional Authentication

Bi-directional authentication is used where a bi-directional link such as a direct connection is available. Bi-directional authentication mode implements an IFF (Identification Friend or Foe) system. In this mode, a 16/32-bit string (the challenge) is sent to the KEELOQ device. In the device, an encryption operation is performed on the challenge, using a 64-bit key. The resultant 16/32-bit string (the response) is then transmitted. This 32-bit string is unique for a given challenge and key. KEELOQ devices may contain several keys.

Transmission Format

The KEELOQ devices use a 66/67-bit transmission format. A 66-bit transmission is composed of a 32-bit encrypted string, a 28-bit fixed string, 4-bit function code, battery low indicator, and a repeat indicator. A 67-bit transmission is available on high-end encoders and is composed of a 32-bit encrypted string, a 28- or 32-bit fixed string, 4- or 0-bit function code, battery low indicator, and a 2-bit CRC. The fixed string is the serial number of the encoder and remains constant for all transmissions from a particular transmitter. However, the 32-bit encrypted string is unique for each transmission.

© 1997 Microchip Technology Inc.

Transmitter Activation

The encoder has an internal power switch which turns on when any of the button inputs are taken high. When a button is pressed, the encoder is turned on and the inputs are sampled after a debounce delay. The encoder then transmits a code, based on the correct key, synchronization information, and function codes.

If the time-out option is selection, the encoder will shut-down after a complete transmission. This mode prevents battery depletion when a push button is inadvertently activated for an extended period. To activate the device again, the buttons have to be released and activated again.

With 28-bit serial number selected:

Hop code			Serial number			Function	Status/CRC
LSB	32 bits	MSB	LSB	28 bits	MSB	4 bits	2/3 bits

With 32-bt serial number selected:

Hop code			Serial number			Status/CRC
LSB	32 bits	MSB	LSB	32 bits	MSB	2/3 bits

Three data rates are used: 833, 1667, and 3333 bits per second. At the lower transmission rate, a complete transmission requires approximately 100ms, at the higher rate 50ms, and at the highest rate 25ms. All encoders will transmit the codes repeatedly while the transmitter is activated.

© 1997 Microchip Technology Inc.

Special Features

Code Word Completion

Code word completion is an automatic feature that makes sure that the entire code word is transmitted, even if the button is released before the transmission is complete. A KEELOQ encoder powers itself up when a button is pushed and powers itself down after the command is finished, if the user has already released the button. If the button is held down beyond the time for one transmission, then multiple transmissions will result. If another button is activated during a transmission, the active transmission will be aborted and the new code will be generated using the new button information.

Blank Alternate Code Word

Federal Communications Commission (FCC) part 15 rules specify the limits on fundamental power and harmonics that can be transmitted. Power is calculated on the worst case average power transmitted in a 100ms window. It is therefore advantageous to minimize the duty cycle of the transmitted word. This can be achieved by minimizing the duty cycle of the individual bits and by blanking out consecutive words. Blank Alternate Code Word (BACW) is used for reducing the average power of a transmission. This is a selectable feature that is determined in conjunction with the baud rate selection bits. Using the BACW allows the user to transmit a higher amplitude transmission if the transmission length is shorter. The FCC puts constraints on the average power that can be transmitted by a device, and BACW effectively prevents continuous transmission by only allowing the transmission of every second or every fourth code word. This reduces the average power transmitted and hence, assists in FCC approval of a transmitter device.

Envelope Encryption Option

Envelope Encryption is a user selectable option which is meant to offer a higher level of security for a code hopping system. During a normal transmission with the envelope encryption turned off, the 28-bit serial number and 4-bit function code is transmitted in the clear (unencrypted). If envelope encryption is selected, then the serial number is also encrypted before transmission. The encryption for the serial number and 4-bit function code is done using a different algorithm than the transmission algorithm. The envelope encryption scheme is not nearly as complex as the KEELOQ algorithm and, hence, not as secure. When the envelope encryption is used, the serial number must be decrypted using the envelope key and envelope decryption. After the serial number is obtained, the normal decryption method can be used to decrypt the hopping code. All transmitters in a system must use the same envelope key.

Auto-shutoff

The Auto-shutoff function automatically stops the device from transmitting if a button inadvertently gets pressed for a long period of time. This will prevent the device from draining the battery if a button gets pressed while the transmitter is in a pocket or purse. This function can be enabled or disabled and is selected by setting or clearing the Auto-shutoff bit. Time-out period is approximately 25 seconds.

VLOW: Voltage LOW indicator

The VLOW bit is transmitted with every transmission and will be transmitted as a one if the operating voltage has dropped below the low voltage trip point. The trip point is selectable between two values, based on the battery voltage being used. This VLOW signal is transmitted so the receiver can give an indication to the user that the transmitter battery is low.

© 1997 Microchip Technology Inc.

DECODER FUNCTIONS

General Description

The KEELOQ decoders can be used for secure remote control systems. To gain a full understanding of the modes of KEELOQ decoder operation, several concepts have to be understood.

Decoder Features

Output Activation

The function information contains information on the buttons that were activated. The function code is embedded in the encrypted portion of the transmission and is also transmitted in the clear if selected. The decoder will activate the switch outputs corresponding to the switch inputs on the encoder as long as the transmitter is activated. A minimum activation time of 500ms is used to allow for dropout in the signal. Buttons are usually assigned to different functions. In a vehicle alarm system one button might be used for arming and locking, another for disarming and unlocking and a third for trunk release. If any button is activated for more than say 2 seconds a panic siren can be activated.

Key Management

For a decoder to respond to an encoder's transmissions, the decoder has to have access to several items of information:

- The *serial number* which is unique for each transmitter. KEELOQ encoders feature 28- or 32-bit serial numbers.
- The *synchronization history* of that transmitter. This information is used to determine whether the transmission is a valid one, or whether it has been transmitted by a code grabber. The form in which this history is maintained makes it possible to re-synchronize the decoder with the transmitter, even if the transmitter has been pressed repeatedly out of range of the receiver.
- The *buttons* used to activate the transmitter.
- The *key* being used by that transmitter. All KEELOQ devices use 64-bit keys. This information is essential to enable the decoder to decrypt the transmission and gain access to the information contained in the encrypted code portion.
  - Single key system
    In a single key system the decoder stores a single key and uses that key for interpreting the transmissions from all transmitters. It follows that all transmitters to be used with that decoder, must be set up with the same key. A single key system is not very secure but the program is simplified and the storage requirements are less.
    The serial number is used to distinguish between different transmitters. The decoder still has to maintain separate synchronization information for each transmitter. This information is maintained in a separate location in the EEPROM memory for each transmitter. The history record for a particular transmitter is updated every time a valid transmission is received, enabling the decoder to keep track of each transmitter independently. Even if one transmitter is used regularly while another is stored away (with or without power), both transmitters will work the first time when activated.
  - Independent key system
    An independent key system is much more secure as the preferred method and is used in all KEELOQ decoders. In independent key systems, the decoder maintains a separate key for each transmitter. The serial number or derivative is used to locate the correct key. Once the correct location has been determined, the key is retrieved and used to decrypt the 32-bit encrypted portion. The correct history record is then retrieved and used to determine whether a valid code has been received.

Learning

Learning is a feature of KEELOQ decoders that allows the addition of new transmitters to the system without having to reprogram the system from outside.

The decoder's learning capability simplifies replacement of lost transmitters. When a transmitter is lost, the user can "teach" the decoder the key of a new transmitter. The previous transmitter's key will be written over, thereby excluding that transmitter from the system.

Once the self learning procedure has been followed, the receiver has learned the new transmitter's identity and (if applicable) the output functions that have to be activated for that transmitter. The new transmitter may then be used normally to activate the decoder.

© 1997 Microchip Technology Inc.

Normal learning mode (Serial Number Derived)

- Transmission format in normal learn mode.

With 28-bit serial number selected:

Hop Code			Serial Number			Function	Status/CRC
LSB	32 bits	MSB	LSB	28 bits	MSB	4 bits	2/3 bits

With 32- bit serial number selected:

Hop Code			Serial Number			Status/CRC
LSB	32 bits	MSB	LSB	32 bits	MSB	2/3 bits

Operation

The user places the decoder in learning mode. When the first code word is received, the serial number is used to generate the transmitter's secret key that was programmed into the transmitter during the production stage. The secret key is used to decrypt the hopping code. The decoder then waits for a second transmission (must be activated a second time). The serial number is compared to the first received serial number. If equal the hopping code is decrypted and the validation checks performed. The final check is to check that the codes are sequential. All the transmitter information is then stored.

Secure learn mode (Seed Derived)

- Transmission format in secure learn mode

With 32 bit serial number selected:

Seed			Serial number			Function	Status/CRC
LSB	32 or 48 bits	MSB	LSB	28 or 12 bits	MSB	4 bits	2/3 bits

With 32 bit serial number selected:

Seed			Serial number			Status/CRC
LSB	32 or 48 bits	MSB	LSB	32 or 24 bits	MSB	2/3 bits

Operation

The user places the decoder in learning mode. During the first stage of learn, the user must press a specific button. The hopping code part of the transmission will be replaced by the 32/48-bit seed stored in the encoder. The key is derived from the seed instead of the serial number in this case. During the second stage of learning, any other button combination can be pressed. A normal hopping code transmission is sent during the second stage. The generated key is used to decrypt the hopping code, and all the transmitter information is stored.

Since the seed is only transmitted during the learning process, and is required to generate the key a normal transmission cannot be intercepted, a key generated and the hopping code decrypted to predict the next hopping code.

As a further security measure, the transmission of the random seed can be disabled after 1 to 128 operations of the transmitter once the transmitter has been learned. That means that even if someone had physical access to the encoder, it would not be possible to get the random seed which is needed for key generation. The disadvantage is that a transmitter cannot be relearned at a later stage.

© 1997 Microchip Technology Inc.

Memory Requirements

A code hopping system typically requires nonvolatile memory such as an EEPROM to store transmitter information. This allows information to be changed when a transmitter is learned and operated but ensures that the information will not be lost if power is removed. Since the receiver needs to store information on each transmitter, the number of transmitters is determined by the available memory size. A typical system requires 16 bytes to be stored for each transmitter.

Memory map:

Transmitter	
1	Learned
2	Learned
3	← Next learn position
~	
n	

A learning pointer can be used to point to the next available learning position. Once position n is reached, the pointer will wrap to position 1.

Transmitter Replacement

The replacement of transmitters is a system implementation dependent. All transmitters can be erased by activating the learn input for 10 seconds. The wanted transmitters can then be relearned. Another method is to use a rotating buffer. Transmitters will be learned into sequential positions until the last position is reached. The next learn will then overwrite the first transmitter.

© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

Automatic Calibration of the WDT Time-out Period

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

The WDT timer is a simple RC timer with a typical time-out period of about 18 ms. This time-out period is dependent on Voltage, Temperature and Silicon process variations. Hence the tolerance on the time-out period is very wide: Min. of 9 ms to a Max. of 33 ms (please refer to appropriate datasheet for device dependent value). There are applications where an additional timer would be useful as an approximate time keeper, hence getting a more precise value of the WDT time-out is useful. This Tech Brief implements an automatic calibration of the WDT time-out period on start-up.

IMPLEMENTATION

The hardware used for this brief is the PICDEM1 board. It is assumed that the main processor oscillator is an accurate crystal or ceramic resonator running at 4 MHz. The program flowchart is depicted in Figure 1. A check is made to see if a certain **Codebyte** exists, if it does not then a power-up is assumed and the calibration is executed. Note that instead of using the 8-bit wide **CodeByte**, the $\overline{PD}$ bit in the **STATUS** register could also be used for this very same purpose. The calibration uses an internal timed 1 ms interval. To get a better resolution, the WDT is postscaled by 4 in order to measure a longer time period . Every 1 ms interval is kept track by incrementing the **WDTValue**. The WDT will eventually time-out and cause a reset. The value in **WDTValue** in then divided by 4 to get the exact WDT time-out period and displayed as a binary number on PORTB. In order to check for repeatability, just press the reset button on the PICDEM1 board and the whole calibration process will be repeated. A power-down, followed by a power-up will also have the same effect.

CONCLUSION

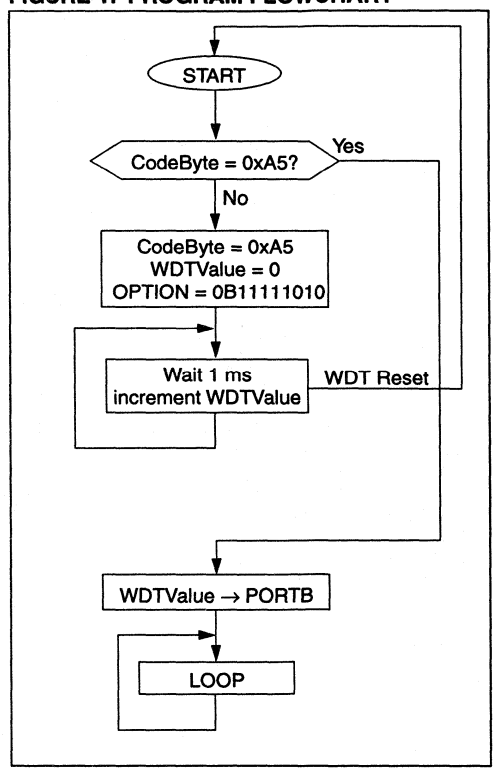
The calibration of the WDT is simple and takes very little overhead in a program. The code in Appendix A is written for a PIC16C84 device, but can be translated to work on any PIC16CXXX product. The code in Appendix B is written for the PIC16C5X family.

RAM Used: 2 Bytes

ROM Used: 50 Words

Execution time: Up to a max. of 132 ms from start-up.

FIGURE 1: PROGRAM FLOWCHART



9

© 1997 Microchip Technology Inc.

APPENDIX A:

```
MPC "C" COMPILER BC.193 22-Aug-1995                                PAGE  1

                                /* In this example, a PIC16C84s WDT is calibrated on startup
                                   for better accuracy.  Accuracy is exact to +/- 0.25mS.
                                   The code written in C works on a PICDEM1 board.  The WDT
                                   value is written to PORTB at the end of the calibration.
                                   The value is in binary. */

                                #pragma option v
                                #include <c:\mpc\16c84.h>
                                #pragma option +1;
03FB                            #define MAXROM 1019
0005 03FB                       #pragma memory ROM [MAXROM]     @ 0x05;
000C 0024                       #pragma memory RAM [36]         @ 0x0C;
                                #pragma option +1;

                                #include <c:\mpc\delay14.h>

                                #pragma option +1;

000D                            char WdtValue;
000E                            chr CodeByte;

                                void InitPorts(void)
                                {
0005 1283    BCF    STATUS,RP0      PORTB = 0; //configure PORTB as output
0006 0186    CLRF   PORTB
0007 1683    BSF    STATUS,RP0      TRISB = 0;
0008 0186    CLRF   PORTB
0009 0008    RETURN               }

                                void main(void)
                                {
000A 2005    CALL   0005h            InitPorts();
000B 30A5    MOVLW  A5h              if(CodeByte != 0xA5)
000C 1283    BCF    STATUS,RP0
000D 0212    SUBWF  TMR1L,W
000E 1903    BTFSC  STATUS,Z
000F 281E    GOTO   001Eh
0010                                   {
0010 1283    BCF    STATUS,RP0       WdtValue = 0;
0011 0191    CLRF   PIR2
0012 30A5    MOVLW  A5h              CodeByte = 0xA5;
0013 0092    MOVWF  TMR1L
0014 30FA    MOVLW  FAh              OPTION = 0B11111010;  // set WDT post scaler = 4
0015 1683    BSF    STATUS,RP0
0016 0081    MOVWF  TMR0
0017 0064    CLRWDT                  CLRWDT();
                                     while(1)
                                       {
0018 3001    MOVLW  01h
0019 202A    CALL   002Ah
001A                                     Delay_Ms_4MHz(1);     // delay for 1mS

001A 1283    BCF    STATUS,RP0        WdtValue++;        // count mS
001B 0A91    INCF   PIR2
001C 2818    GOTO   0018h             }
                                     }
001D 2827    GOTO   0027h          else
                                     {
001E 1283    BCF    STATUS,RP0       CodeByte = 0;
001F 0192    CLRF   TMR1L
```


© 1997 Microchip Technology Inc.

```
0020 0811    MOVF   PIR2,W                PORTB = WdtValue >> 2;     // divide by 4 for
                                                                    ;     // exact timeout value
0021 008D    MOVWF  PIR1
0022 0C8D    RRF    PIR1
0023 0C8D    RRF    PIR1
0024 303F    MOVLW  3Fh
0025 050D    ANDWF  PIR1,W
0026 0086    MOVWF  PORTB
                                        }
                                     while(1)
0027 0064    CLRWDT                           CLRWDT();          // loop forever
0028 2827    GOTO   0027h
0029 0008    RETURN                        }

                                     void Delay_Ms_4MHz(registerw delay)
0000                                 {
                                     #asm
002A 1283                           BCF STATUS, RP0      ;1
002B 008D                           MOVWF __WImage       ;1

                            DLMS4M1
                                    RADIX DEC  ;Use decimal values
002C 30F9                           MOVLW 249            ;1
002D 0084                           MOVWF FSR            ;1

                            DLMS4M2                      ; 4 cycles
002E 0000                           NOP                  ;1
002F 0B84                           DECFSZ  FSR          ;1
0030 282E                           GOTO    DLMS4M2      ;2

0031 0B8D                           DECFSZ __WImage      ;1
0032 282C                           GOTO DLMS4M1         ;2
                                           #endasm
0033 0008    RETURN                        }

0000 3000    MOVLW  00h
0001 008A    MOVWF  PCLATH
0002 280A    GOTO   000Ah


ROM USAGE MAP

    0000 to 0002    0005 to 0033
    Total ROM used 0032


Errors              :    0
Warnings            :    0
```

9

APPENDIX B:

```
MPC "C" COMPILER BC.193 22-Aug-1995                                  PAGE  1

                                /* In this example, a PIC16C54s WDT is calibrated on startup
                                 for better accuracy. Accuracy is exact to +/- 0.25 mS.
                                 The code written in C works on a PICDEM1 board. The  WDT
                                 value is written to PORTB at the end of the calibration.
                                 The value is in binary. */

                                #pragma option v
                                #include <c:\mpc\16c54.h>
                                #pragma option +l;  // Enable header to show branch islands
0200                            #define MAXROM 512
0000 0200                       #pragma memory ROM [MAXROM] @ 0x00;
0008 0018                       #pragma memory RAM [24]     @ 0x08;  // Reserve byte for
                                                                    ;  //  __WImage
                                #pragma option +l;

                                #include <c:\mpc\delay12.h>

                                #pragma option +l;


0009                            char WdtValue;
000A                            char CodeByte;


                                void InitPorts(void)
                                {
0000 0066   CLRF   PORTB            PORTB = 0;      //set PORTB as output
0001 0C00   MOVLW  00h              __TRIS(0,PORTB);
0002 0006   TRIS   PORTB
0003 0800   RETLW  00h          }

                                void main(void)
                                {
0004 0900   CALL   0000h            InitPorts();
0005 0CA5   MOVLW  A5h              if (CodeByte != 0xA5)
0006 008A   SUBWF  0A,W
0007 0643   BTFSC  STATUS,Z
0008 0A14   GOTO   0014h
0009                                  {
0009 0069   CLRF   09                 WdtValue = 0;
000A 0CA5   MOVLW  A5h                CodeByte = 0xA5;
000B 002A   MOVWF  0A
000C 0CFA   MOVLW  FAh                __OPTION(0B11111010); // set WDT post scaler = 4
000D 0002   OPTION
000E 0004   CLRWDT                    CLRWDT();
                                      while(1)
                                        {
000F 0C01   MOVLW  01h
0010 091F   CALL   001Fh
0011                                    Delay_Ms_4MHz(1); // delay for 1 mS
0011 02A9   INCF   09                   WdtValue++;       // count 1ms delays
0012 0A0F   GOTO   000Fh                }
                                      }
0013 0A1C   GOTO   001Ch            else

                                      {
0014 006A   CLRF   0A                 CodeByte = 0;
0015 0209   MOVF   09,W               PORTB = WdtValue >> 2;  //divide by 4 for exact
                                                              // timeout value
0016 0027   MOVWF  PORTC
0017 0327   RRF    07
```


© 1997 Microchip Technology Inc.

```
0018 0327    RRF    07
0019 0C3F    MOVLW  3Fh
001A 0147    ANDWF  07,W
001B 0026    MOVWF  PORTB
                                    }
                                 while(1)
001C 0004    CLRWDT                CLRWDT(); // loop forever
001D 0A1C    GOTO   001Ch
001E 0800    RETLW  00h          }


                                 void Delay_Ms_4MHz(registerw delay)
0000                             {
                                 #asm
001F 0000                  NOP     ;1
0020 0027                  MOVWF __WImage      ;1

                         DLMS4M1
                            RADIX DEC  ;Use decimal values
0021 0CF9                   MOVLW 249          ;1
0022 0028                   MOVWF __FSRImage          ;1

                         DLMS4M2               ;4 cycles
0023 0000                     NOP               ;1
0024 02E8                     DECFSZ __FSRImage       ;1
0025 0A23                     goto  DLMS4M2     ;2

0026 02E7                   DECFSZ __WImage     ;1
0027 0A21                   goto DLMS4M1        ;2
                                 #endasm
0028 0800    RETLW  00h          }

01FF 0A04    GOTO   0004h


ROM USAGE MAP

    0000 to 0028    01FF to 01FF
    Total ROM used 002A


Errors              :    0
Warnings            :    0
```

9

NOTES:

© 1997 Microchip Technology Inc.

Using External RAM with PIC17CXXX Devices

Author: *Rodger Richey*
Microchip Technology inc.

INTRODUCTION

This Technical Brief shows how to connect a PIC17CXXX device to external memory. It also provides instructions and calculations to help determine which speeds of SRAM work with which frequency crystal.

SYSTEM CONFIGURATION

Are you doing FFT's? Storing arrays of data? Typically, low speed designs would use EEPROM to hold the data that is being generated. On the other hand, high speed designs need to use static RAM because of the faster access times. External SRAM can easily be integrated with a PIC17CXXX device to create the SRAM bank your design requires. Data storage and retrieval is accomplished through table read and table write instructions.

All the equations for lead timing can also be applied to EPROMs.

The first thing you need to know is how much RAM the system requires. Table 1 shows how much external memory can be used with the PIC17CXXX family members. The microcontroller is put into extended microcontroller mode which can access both the internal and external memory. Both the internal and external memory is 16-bits wide.

TABLE 1: INTERNAL/EXTERNAL MEMORY

Device		Size (Words)	Address Range
PIC17C42	Internal	2048	0x0000 - 0x07FF
	External	63488	0x0800 - 0xFFFF
PIC17C43	Internal	4096	0x0000 - 0x0FFF
	External	61440	0x1000 - 0xFFFF
PIC17C44	Internal	8192	0x0000 - 0x1FFF
	External	57344	0x2000 - 0xFFFF

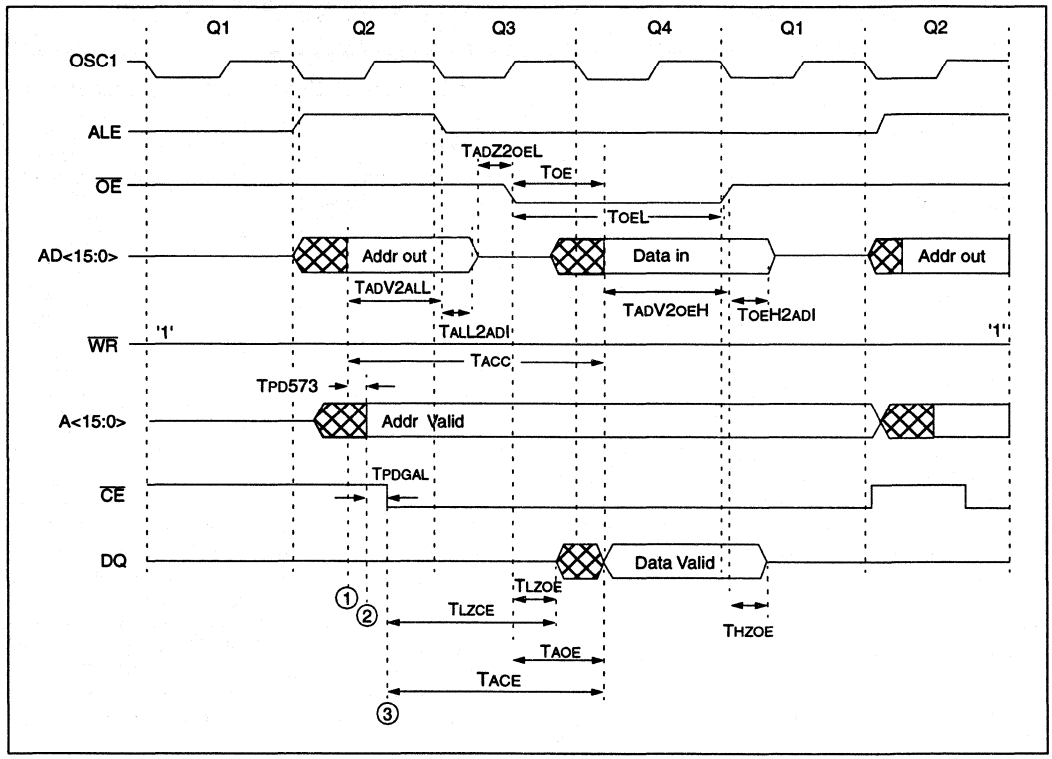
The next step is to connect the external SRAM to the PIC17CXXX. Appendix A shows the schematic for connecting two MT5C2568 (32Kx8) SRAM devices to a PIC17C43. The MT5C2568 can be obtained from Micron Semiconductor. The only additional hardware required is two latches (74ACT373s or 74ACT573s). Both chips are octal latches but the 74ACT573 has the inputs and the outputs on opposite sides of the chip which makes interfacing to the microcontroller easier.

© 1997 Microchip Technology Inc.

READ TIMINGS

Now that we know how to connect the hardware, the next step is to determine the speed grade of the SRAM. Let's first take a look at the read timings. Figure 1 shows a combined timing diagram for the read cycle of a PIC17C4X device and the MT5C2568 SRAM.

FIGURE 1: PIC17C4X/MT5C2568 READ CYCLE TIMING DIAGRAM



The value of TACE shows which speed grade of SRAM is required for the read cycle. Figure 1 has three key locations marked that help define TACE. Table 2 summarizes these locations.

TABLE 2: LOCATION MARKER DESCRIPTIONS FOR FIGURE 1

Location	Description	Value	
1	Address Setup Time. This location marks the spot where the address becomes valid out of the PIC17CXXX before ALE goes LOW. (Q2 cycle)	0.25 * TCY - 10	time in nanoseconds before the Q3 cycle (TADV2ALL)
2	Propagation delay of the 74ACT573. This is the spot where the address to the SRAM becomes valid. (Q2 cycle)	6	time in nanoseconds before the Q3 cycle (TPD573)
3	Propagation delay of the address decoder. This spot is where the CE signal from the address decoder to the SRAM goes LOW. (Q2 cycle)	TPDGAL	time in nanoseconds before the Q3 cycle

© 1997 Microchip Technology Inc.

TABLE 3: MEMORY INTERFACE READ REQUIREMENTS

Sym	Characteristic	Min	Typ	Max	Units
TADV2ALL	AD15:AD0 (address) valid to ALE↓ (address setup time)	0.25Tcy - 10	—	—	ns
TALL2ADI	ALE↓ to address out invalid (address hold time)	5*	--	—	ns
TADZ2OEL	AD15:AD0 hi-impedance to $\overline{OE}$↓	10	—	—	ns
TADV2OEH	Data in valid before $\overline{OE}$↑ (data setup time)	35	—	—	ns
TOEH2ADI	$\overline{OE}$↑to data in invalid (data hold time)	0	—	—	ns
TOEL	$\overline{OE}$ pulse width	0.5Tcy - 35	—	—	ns
TOE	Output enable access time ($\overline{OE}$ low to Data Valid)	—	—	0.5TCY - 45	ns
TPD573	Propagation delay input to output	2	6	12	ns
TACC	Address access time	0.75Tcy - 30	—	—	ns
TPDGAL	Propagation delay input to combinatorial output	3	—	7.5 - 25	ns
TLZOE	$\overline{OE}$ to output in Low - Z	0	—	—	ns
TAOE	$\overline{OE}$ access time	—	—	5 - 8	ns
TLZCE	$\overline{CE}$ to output in Low - Z	3	—	—	ns
TACE	$\overline{CE}$ access time	—	—	10 - 25	ns
THZOE	Output disable to output in High - Z	—	—	5 - 7	ns

To determine TACE from Figure 1, the following equation can be used. Each part of the equation is related to a timing parameter or one of the marked locations in Figure 1.

TACE =	0.75 * TCY - 30	- Address access time - TACC
	- 6	- Propagation delay of the 74ACT573 - TPD573
	- TPDGAL	- Propagation delay of the address decoder

Finally, lets find some example speed grades for various crystal frequencies. One thing to remember is that this is only the read timing speed grade. You must also calculate the write timing speed grade. As you can see the speed grade for the SRAM is dependent on the address decoder. Typically, programmable array logic devices such as a 16V8 or a 22V10 are used. The propagation delays of these devices can be anywhere from 5 ns up to 30 ns.

8 MHz (TCY = 500 ns)	**16 MHz (TCY = 250 ns)**	**20 MHz (TCY = 200 ns)**	**25 MHz (TCY = 160 ns)**
TACE = 0.75 * 500 - 30 - 6 - TPDGAL	TACE = 0.75 * 250 - 30 - 6 - TPDGAL	TACE = 0.75 * 200 - 30 - 6 - TPDGAL	TACE = 0.75 * 160 - 30 - 6 - TPDGAL
TACE = 345 - 6 - TPDGAL	TACE = 157.5 - 6 - TPDGAL	TACE = 125 - 6 - TPDGAL	TACE = 90 - 6 - TPDGAL
TACE = 339 - TPDGAL	TACE = 151.5 - TPDGAL	TACE = 119 - TPDGAL	TACE = 84 - TPDGAL

To reduce the total propagation delay of the 74ACT573 and the address decoder, the address/data lines and ALE from the PIC17C4X can be routed to the address decoder. Now the address latches and the address decoding can be performed in parallel.

9

© 1997 Microchip Technology Inc.

WRITE TIMINGS

Now that you have determined the read timing speed grade, it is time to calculate the write timing speed grade of the SRAM for your design. Figure 2 shows the combined timings for a write cycle of the PIC17C4X devices and the MT5C2568. Again, there are markings for some key locations which are described in Table 4.

FIGURE 2: PIC17C4X/MT5C2568 WRITE CYCLE TIMING DIAGRAM

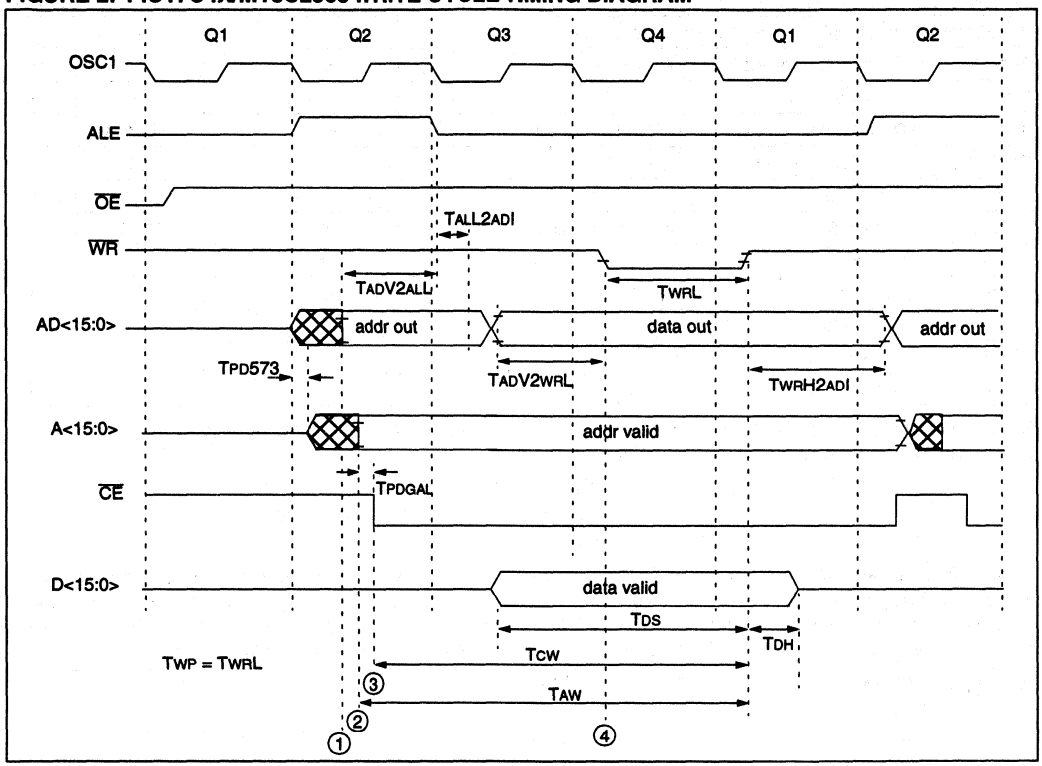


TABLE 4: LOCATION MARKER DESCRIPTIONS FOR FIGURE 3

Location	Description	Value	
1	Address Setup Time. This location marks the spot where the address becomes valid out of the PIC17CXXX before ALE goes LOW. (Q2 cycle)	0.25 * TCY - 10	time in nanoseconds before the Q3 cycle (TADV2ALL)
2	Propagation delay of the 74ACT573. This is the spot where the address to the SRAM becomes valid. (Q2 cycle)	6	time in nanoseconds before the Q3 cycle
3	Propagation delay of the address decoder. This spot is where the CE signal from the address decoder to the SRAM goes LOW. (Q2 cycle)	TPDGAL	time in nanoseconds before the Q3 cycle
4	Write pulse goes LOW. The SRAM specifies that the address at it's inputs must be valid at or before this point. (Q4 cycle)		

© 1997 Microchip Technology Inc.

TABLE 5: MEMORY INTERFACE WRITE REQUIREMENTS

Sym	Characteristic	Min	Typ	Max	Units
TADV2ALL	AD<15:0> (address) valid to ALE↓ (address setup time)	0.25Tcy - 10	—	—	ns
TALL2ADI	ALE↓ to address out invalid (address hold time)	0	—	—	ns
TADV2WRL	Data out valid to $\overline{WR}$↓ (data setup time)	0.25Tcy - 40	—	—	ns
TWRH2ADI	$\overline{WR}$↑ to data out invalid (data hold time)	—	0.25TCY	—	ns
TWRL	$\overline{WR}$ pulse width	—	0.25TCY	—	ns
TPD573	Propagation delay input to output	2	6	12	ns
TPDGAL	Propagation delay input to combinatorial output	3	—	7.5 - 25	ns
TDS	Data setup time	6 - 10	—	—	ns
TCW	$\overline{CE}$ to end of write	12 - 15	—	—	ns
TAW	Address valid to end of write	7 - 15	—	—	ns
TDH	Data hold time	0	—	—	ns
TWP	$\overline{WE}$ pulse width Low	7 - 15	—	—	ns

The critical timing specification for the write cycle is when the CE signal from the address decoder to the SRAM goes LOW. Typically the SRAM specifies this parameter (TAS for the MT5C2568). The other important SRAM specifications to check against the PIC17C4X are:

- TWP - Write enable LOW pulse width
- TDS - Data setup time, data valid before WE goes HIGH
- TDH - Data hold time, data valid after WE goes HIGH

The following are some calculations for the write cycle of various speed grades. These calculations are used to determine if any of the above specifications are invalid. The Micron MT5C2568 specifies the following values for the above parameters:

TAS - 0 ns min. for all speed grades

TWP - 15 ns min. for the 25 ns speed grade

TDS - 10 ns min. for the 25 ns speed grade

TDH - 0 ns min. for all speed grades

The PIC17C4X specifies the following values:

TWP - 0.25 * TCY	TWRL
TDS - 0.25 * TCY - 40 + TWP min.	TADV2WRL
TDH - 0.25 * TCY	TWRH2ADI

9

© 1997 Microchip Technology Inc.

8 MHz (TCY = 500 ns)	16 MHz (TCY = 250 ns)	20 MHz (TCY = 200 ns)	25 MHz (TCY = 160 ns)
TADV2ALL = 0.25 * 500 - 10	TADV2ALL = 0.25 * 250 - 10	TADV2ALL = 0.25 * 160 - 10	TADV2ALL = 0.25 * 160 - 10
TAS = TADV2ALL	TAS = TADV2ALL	TAS = TADV2ALL	TAS = TADV2ALL
- 6	- 6	- 6	- 6
- TPDGAL	- TPDGAL	- TPDGAL	- TPDGAL
+ TCY	+ TCY	+ TCY	+ Tcy
TWRL = 0.25 * 500	TWRL = 0.25 * 250	TWRL = 0.25 * 160	TWRL = 0.25 * 160
TADV2WRL = 0.25 * 500 - 40 + 0.25 * 500	TADV2WRL = 0.25 * 250 - 40 + 0.25 * 250	TADV2WRL = 0.25 * 160 - 40 + 0.25 * 160	TADV2WRL = 0.25 * 160 - 40 + 0.25 * 160
TWRH2ADI = 0.25 * 500	TWRH2ADI = 0.25 * 250	TWRH2ADI = 0.25 * 160	TWRH2ADI = 0.25 * 160
TADV2ALL = 115	TADV2ALL = 52.5	TADV2ALL = 40	TADV2ALL = 30
TAS = 95	TAS = 32.5	TAS = 20	TAS = 10
- 6	- 6	- 6	- 6
- TPDGAL	- TPDGAL	- TPDGAL	- TPDGAL
+ 125	+ 62.5	+ 50	+ 40
TWRL = 125	TWRL = 62.5	TWRL = 50	TWRL = 40
TADV2WRL = 210	TADV2WRL = 227.5	TADV2WRL = 60	TADV2WRL = 40
TWRH2ADI = 125	TWRH2ADI = 62.5	TWRH2ADI = 50	TWRH2ADI = 40
TADV2ALL = 115	TADV2ALL = 52.5	TADV2ALL = 40	TADV2ALL = 30
TAS = 214 - TPDGAL	TAS = 89 - TPDGAL	TAS = 64 - TPDGAL	TAS = 44 - TPDGAL
TWRL > 15 ns	TWRL > 15 ns	TWRL > 15 ns	TWRL > 15 ns
TADV2WRL > 15 ns	TADV2WRL > 15 ns	TADV2WRL > 15 ns	TADV2WRL > 15 ns
TWRH2ADI > 0 ns	TWRH2ADI > 0 ns	TWRH2ADI > 0 ns	TWRH2ADI > 0 ns

Assuming that a 16V8 with propagation delay of 25 ns is used for the address decoder, you can see that all crystal frequencies have a TAS greater than 0. The other parameters TADV2WRL, TWRH2ADI, and TWRL all have at least 25 ns of spare room. The parameters TADV2WRL, TWRH2ADI, and TWRL were taken from a 25 ns speed grade SRAM. Therefore, all the speed grades of the MT5C2568 can be used for the write cycle of a PIC17C4X with an external oscillator frequency up to 25 MHz.

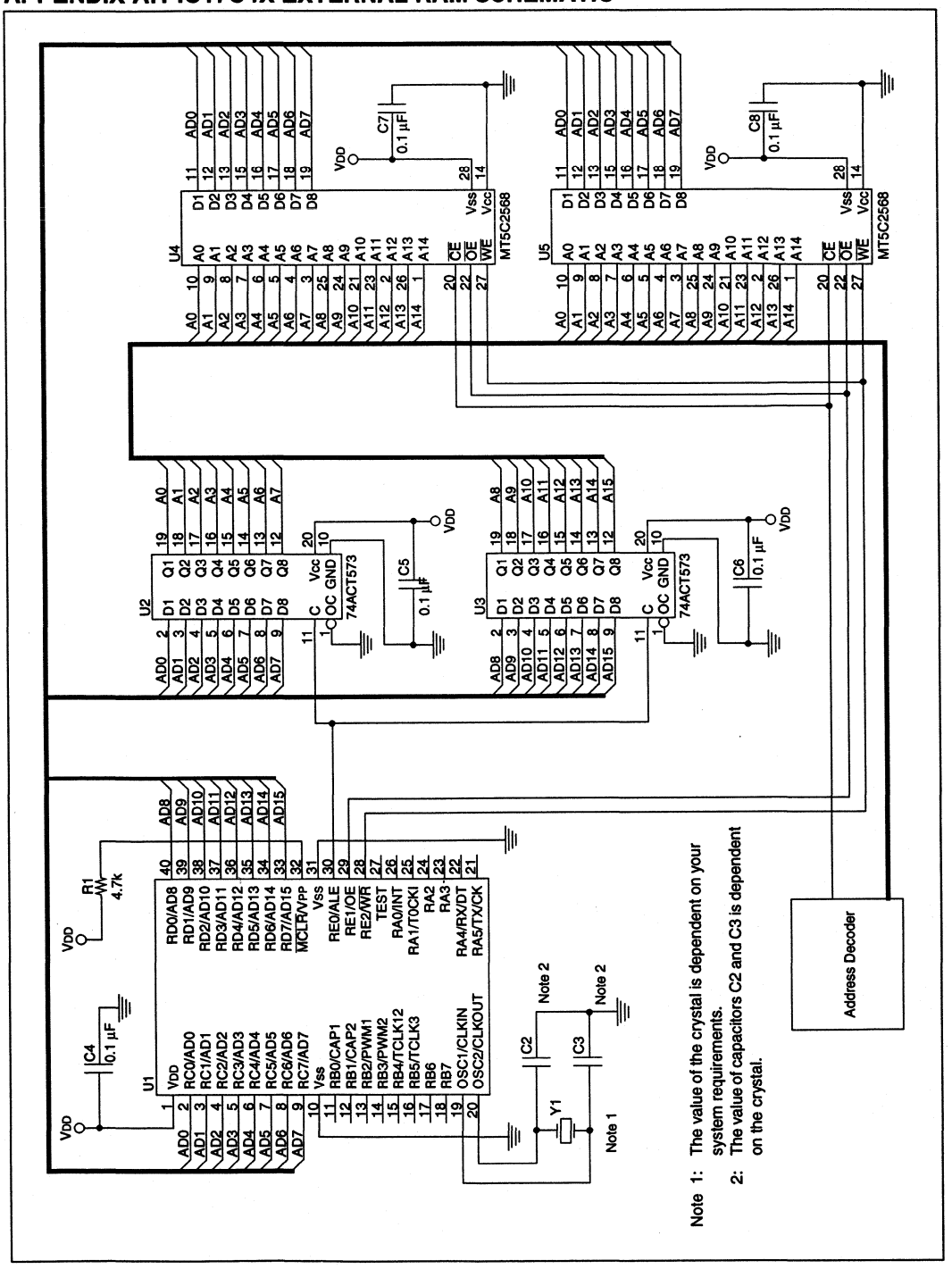
CONCLUSION

So now that we have done the calculations, let's determine the proper SRAM speed grade for the various frequencies. The only assumption is that a 25 ns 16V8 is used for address decoding.

- 8 MHz can use up to a 300 ns SRAM
- 16 MHz can use up to a 125 ns SRAM
- 20 MHz can use up to a 90 ns SRAM
- 25 MHz can use up to a 50 ns SRAM

© 1997 Microchip Technology Inc.

APPENDIX A: PIC17C4X EXTERNAL RAM SCHEMATIC



9

NOTES:

© 1997 Microchip Technology Inc.

Transformerless Power Supply

Author: *Stan D'Souza*
Microchip Technology inc.

INTRODUCTION

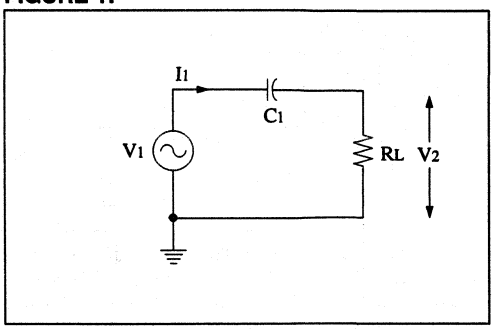
In most non-battery applications, the power to the microcontroller is normally supplied using a wall mounted transformer which is then rectified, filtered and regulated. In most applications this method of generating the regulated voltage is cost effective and can be justified. However there are applications where the PIC12/16/17 is the main controller and low voltage is not required by other components except the PIC12/16/17. In these instances the cost of the transformer becomes the sizable cost factor in the system. Transformerless power supplies, thus, have a distinct advantage in cost as well as in size. The disadvantages of using a transformerless power supply are: (1)Low current supply and (2) no isolation from the AC line voltage. The PIC12/16/17 microcontrollers draw a maximum of 10 mA, even at the highest frequency and voltage of operation, therefore low current availability is not an issue. AC line voltage isolation can be addressed by using MOVs or transient suppressors on the PIC12/16/17.

IMPLEMENTATION

When a capacitor and resistor are connected in series to an AC source, as in Figure 1, a constant current can be maintained through the resistor, so long as the reactance of the capacitors is much greater than the resistance. The current flow is dependent upon the value of the capacitor and assuming that V_1 is much greater than V_2, the value of the current can be assumed to be:

$I_{RMS} = V_1/X_C$ where X_C is the reactance of the capacitor.

FIGURE 1:

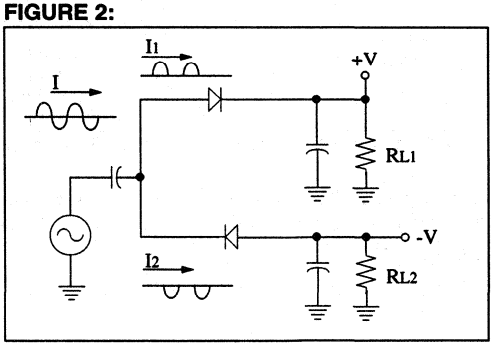


Assuming a line voltage of $V_1 = 115V$ and line frequency = 60 Hz,

$I_{RMS} = 115(2\pi 60C) = 4300C$ or $I_{RMS} \approx 40$ mA/µF.

In order to get a DC voltage using this system, a pair of rectifiers and filter caps can be added as shown in Figure 2. This would give us a capability of driving one half of the current through the positive part and the other half through the negative part. So the max current on each side would be = 20 mA/µF.

FIGURE 2:

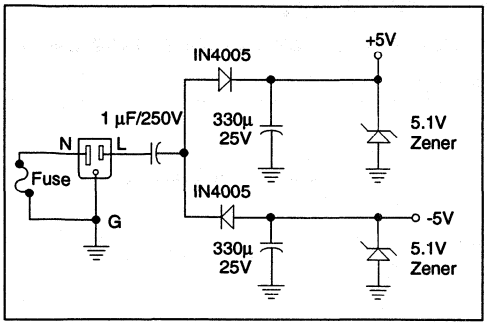


9

© 1997 Microchip Technology Inc.

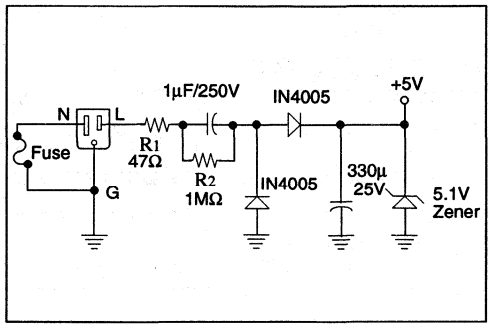
In most applications the output voltage should be regulated. Figure 3 shows a diagram for a practical circuit where a +/- 5V regulated output is generated. Note that the Neutral is connected to ground, through a fuse. This would guard against improper AC wiring.

FIGURE 3:



In most PIC12/16/17 applications the negative voltage is not required. Figure 3 has been thus modified for a single ended supply as in Figure 4. Note that R1 and R2 are required for UL approval.

FIGURE 4:



Precautions:

1. As mentioned earlier, the neutral should be connected to earth ground through a fuse. This would insure protection in case of improper wiring.
2. The AC coupling capacitors and rectifier diodes should be able to withstand the peak voltage in the system. Remember $V_{RMS} = V_{PG}/2$, where $V_{PG} = V_{PEAK}$ to earth ground voltage. Hence for a 120V system the max voltage = 170V. For a 240V system the max voltage = 340V.

TABLE 1: COST ANALYSIS OF A +5V POWER SUPPLY WITH/WITHOUT A TRANSFORMER:

Components	Unit Price	Qty	Transformer P.S.	Qty.	Transformerless P.S.
1.1 VA transformer	5.65	1	5.65	0	-
1N4005 diode	0.05	2	0.10	2	0.10
1 µF/250V Cap	0.57	0	-	1	0.57
5.1V Zener	0.11	1	0.11	1	0.11
330 µF/25V Filter Cap	0.17	1	0.17	1	0.17
Total			6.03		0.95

All Prices based on 100 Qty price from Digikey Catalog.

As can be seen the cost of the transformer power supply is 6 times that of the transformerless power supply. In quantities of 100, the cost of a transformerless power supply is less than a buck.

Conclusion:

A Transformerless power supply is a low cost (under a buck), small footprint alternative by which a PIC12/16/17 can be powered.

© 1997 Microchip Technology Inc.

Adding a Simple 4-channel 8-bit A/D to a PIC17C4X

Author: *Stan D'Souza*
Microchip Technology Inc.

INTRODUCTION

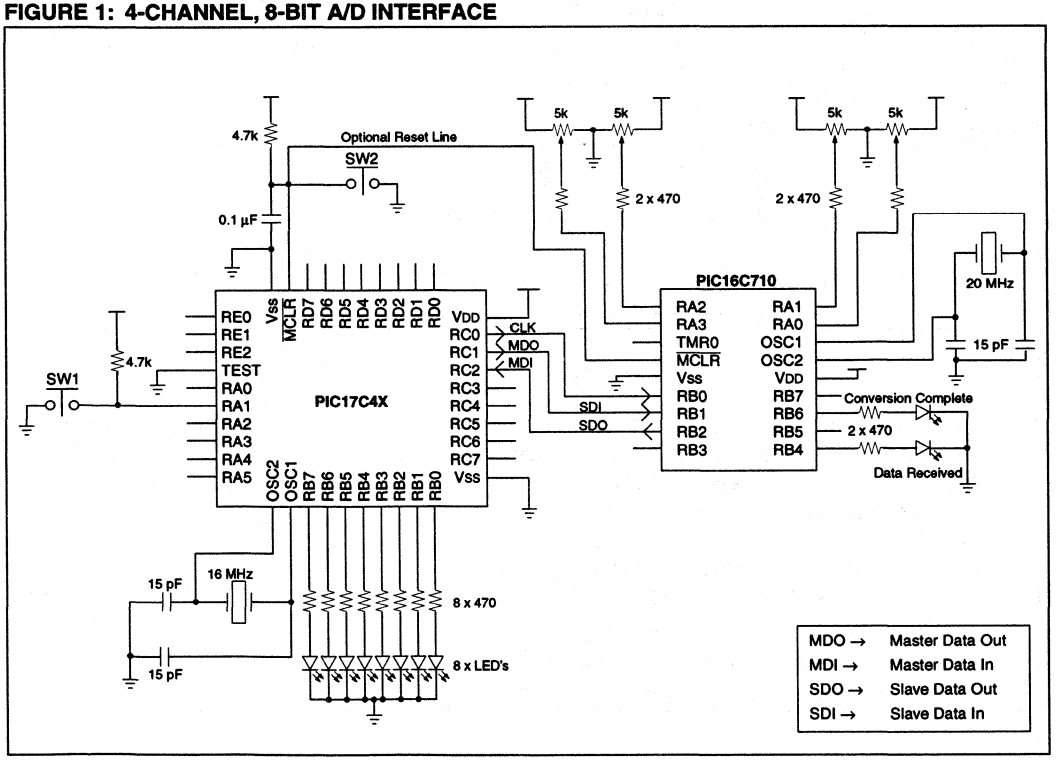
The PIC17C4X high end microcontroller family, offered by Microchip, has many features which make it one of the fastest 8-bit microcontrollers in the industry. However one feature which is missing is an on-chip Analog-to-Digital (A/D) converter. This Technical Brief describes how to connect a PIC16C710 to a PIC17C4X through a simple Serial I/O (SIO) 3 line interface. The result is a quick method by which a user can get a fast (approx. 17 kHz transfer rate) 4-channel A/D converter.

IMPLEMENTATION

The PIC17C4X is the master device in this case, operating at 16 MHz . The master generates the CLK pulse and initiates the conversion. The PIC16C710 is the slave, operating at 20 MHz, and provides the A/D hardware. The two devices are connected using 3 I/O lines, clock, data in, and data out, as shown in Figure 1. The additional connections in Figure 1 are purely for this example and can be omitted if not needed.

The sequence of timed events are depicted in Figure 1. The Master (PIC17C4X) generates the CLK signal and initiates the conversion by transmitting the ADCON register value to the slave (PIC16C710). The slave (PIC16C710) simultaneously transmits ADRES back to the master (PIC17C4X). The ADCON register value has information requesting a new A/D value. The ADRES is the 8-bit result of the last A/D conversion

FIGURE 1: 4-CHANNEL, 8-BIT A/D INTERFACE



9

© 1997 Microchip Technology Inc.

requested. One transmission set consists of 8 clock pulses during which the rising edge of each clock pulse is used to transfer data (ADCON register value) from the PIC17C4X to the PIC16C710 and the falling edge of each clock pulse is used to transfer data (ADRES register value) from the PIC16C710 to the PIC17C4X. This method results in a quick transfer of data back and forth from the two processors. This method of transfer also implies that when the master is requesting for a new A/D value from the slave, it is receiving the A/D result of the last request. The two 8-bit register values (ADCON and ADRES) are maintained in the PIC17C4X and the PIC16C710. The PIC17C4X maintains ADCON and ADRES in general purpose RAM. The PIC16C710 maintains the ADCON in general purpose RAM and the ADRES as a special function register. As seen by its name the ADRES is the register which contains the result of the A/D conversion. ADCON register on the other hand is a merger of various bits from the ADCON0 and ADCON1 special function register defined in the PIC16C710. The definitions for ADCON bits are as below:

ADCON bit #	PIC16C710 definitions	PIC16C710 bit assignments
0	PCFG0	ADCON1<0>
1	PCFG1	ADCON1<1>
2	GO/$\overline{\text{DONE}}$	ADCON0<2>
3	CHS0	ADCON0<3>
4	CHS1	ADCON0<4>
5	Not used	Not Used
6	ADCS0	ADCON0<6>
7	ADCS1	ADCON0<7>

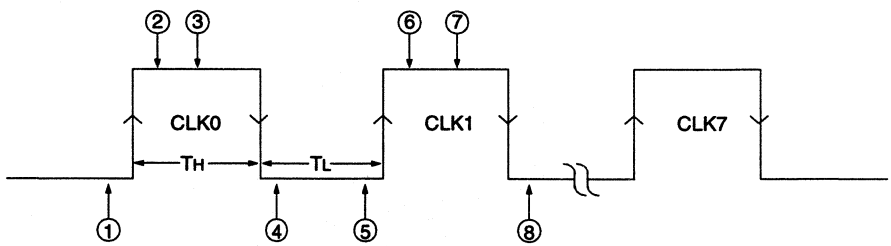
With this setup the user has full control and access to all the A/D configuration features of the PIC16C710. In the example code in Appendix A and Appendix B, the PORTA pins have been all selected as digital I/O and the A/D conversion clock is the internal RC oscillator.

A key feature of this arrangement is the simplicity of the interface. To minimize the number of I/O lines used, the master can reset the slave without having to use the slave's Master Clear ($\overline{\text{MCLR}}$) pin. This can be accomplished by using the PIC16C710's Watchdog Timer (WDT) enabled with a time-out of 18 ms. Once the leading edge of the transfer string is initiated, the master has 8 ms in which to complete the transfer. If the transfer takes longer than the WDT time-out, the WDT will time-out and the PIC16C710 will reset. This feature allows the master to reset the slave, at any time, by initiating a transfer but not completing it before the WDT times out. With a 1:1 prescaler setting, the PIC16C710's WDT will time out between 18 ms and 40 ms.

CONCLUSION

The addition of the PIC16C710 to a PIC17C4X gives the analog input capability to the high-end microcontroller family while only sacrificing 123 words of program memory and 2 bytes of RAM. The simple program in the PIC16C710 uses only 82 words of program memory and 3 bytes of RAM. Since only four pins on the PIC16C710 are used for analog input, the rest of the microcontroller can be used as general purpose I/O if needed.

FIGURE 1: INTERFACE TIMING



1. PIC17C4X sets MDO line = bit0 of ADCON.
2. PIC16C710 gets interrupt and reads bit0 value of ADCON on SDI line.
3. PIC16C710 sets SDO line = bit0 of ADRES.
4. PIC17C4X reads bit0 of ADRES on MDI line.
5. PIC17C4X sets MDO line = bit1 of ADCON.
6. PIC16C710 reads bit1 of ADCON on SDI line.
7. PIC16C710 sets SDO bit = bit1 of ADRES.
8. PIC17C4X reads bit1 of ADRES on MDI line.

Steps 1 through 4 are repeated till all 8-bits are received and sent.

The Clock is generated by the Master.

MDO → Master Data Out
MDI → Master Data In
SDO → Slave Data Out
SDI → Slave Data In
TH > 2 µs min.
TL > 2 µs min.

© 1997 Microchip Technology Inc.

PERFORMANCE

The conversion rate is the addition of the SIO transfer rate as well as acquisition time rate:

The calculation is as follows:

$$[(2 + 2) \bullet 8 + 25]\ \mu s = 57\ \mu s \Rightarrow 17.5\ kHz.$$

FIGURE 2: FLOWCHART FOR MASTER (PIC17C4X)

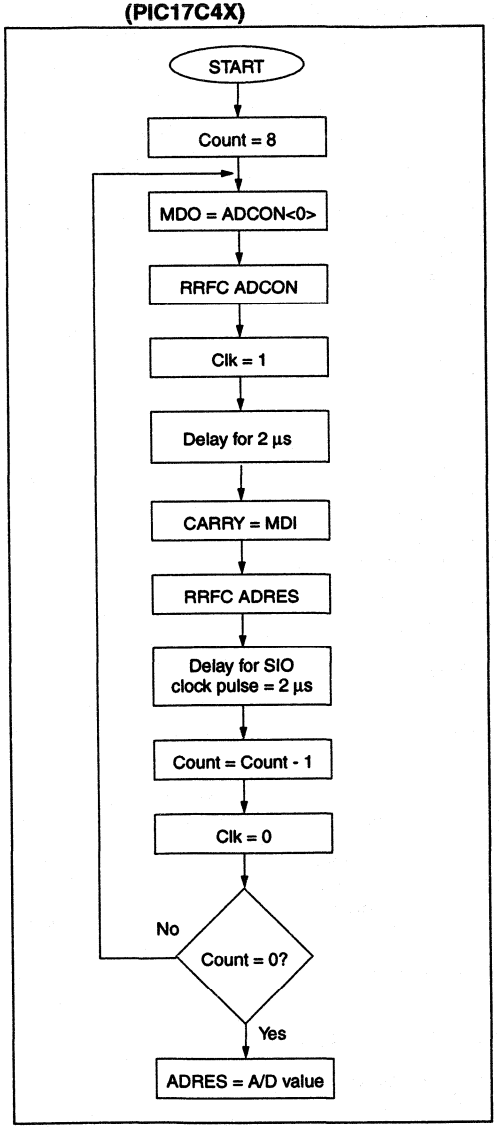
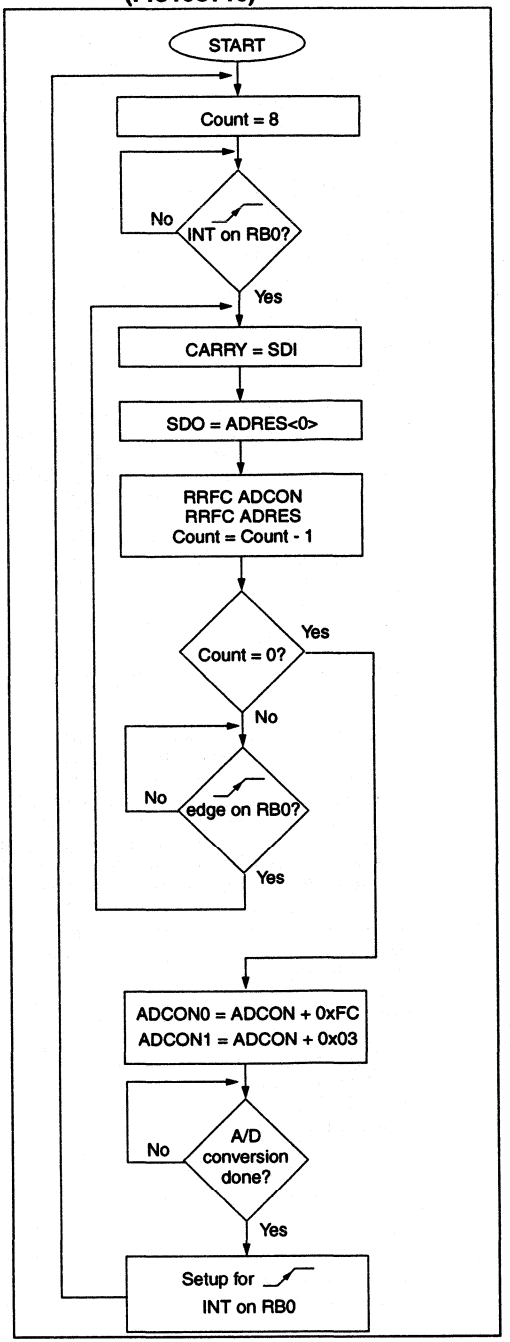


FIGURE 3: FLOWCHART FOR SLAVE (PIC16C710)



9

© 1997 Microchip Technology Inc.

APPENDIX A: CODE FOR PIC17C4X (MASTER)

```
/* This program implements the master control of the ADC interface
   with a PIC16C710 as a simple serial ADC device for use with the
   PIC17C4X.
   The 3 wire interface is as follows:
    CLK --> RC0 output
    DO  --> RC1 output
    DI  --> RC2 input
   The PIC17C4X will transmit an 8-bit byte to the PIC16C710
   in order for it to configure registers ADCON1 and ADCON0 as follows:
    Bit 0 --> PCFG0
    Bit 1 --> PCFG1
    BIT 2 --> Start
    BIT 3 --> CHS0
    BIT 4 --> CHS1
    BIT 5 --> not used
    BIT 6 --> ADCS0
    BIT 7 --> ADCS1
   The ADCON0 and ADCON1 registers are merged into one register
   to help reduce the interface overhead. After all the 8 bits are
   received, bits 0 and 1 are placed in ADCON1 and bits 3 to 7
   are placed in ADCON0.  Depending on the state of bit 2,
   the A/D conversion is then started.

   The protocol basics are as follows:
    Two registers are maintained in the PIC17C4X which are
    used in the transfer:
    ADCON --> ADCON1 merged with ADCON0
    ADRES --> same as the AD result register
    8 clock signals are used to transmit the ADRES value
    and receive the next ADCON value. On the rising edge of
    the clock the serial ADCON value is sent whilst on the
    falling edge of the same clock the serial ADRES value
    is read.  In this manner an efficient transfer
    rate is maintained with very little overhead to software
    and more importantly rate of A/D conversion.
    The PIC16C710 also maintains a replica of ADCON and ADRES
    and does just the opposite of what the PIC17C4X does i.e
    ADCON is received and ADRES is transmitted.  The clock
    master is always the PIC17C4X, the PIC16C710 is always the
    slave.  The transfer occurs as below:
    Clock   Data IN(DI) Data OUT(DO)
    ------------------------------------------------
    Rising edge    N/A       RRF ADCON; C --> DO
    Falling edge   read DI     N/A

    This is repeated till all 8 bits are transmitted/received.

    On receiving the 8-bit value, the ADCON1 and ADCON0 registers
    configured and the A/D conversion started.  The result of the
    conversion is then stored in the ADRES register.  The next
    transfer then sends the ADRES result out.

    NOTE: The PIC17C4X has to provide enough time for sampling
          and for the conversion to complete.

    The WDT on the PIC16C710 is enabled and used in the following manner:

    The entire 8 clock and A/D conversion should be done in a
    max of 8 mS, which is the min. timeout period for the WDT.
    If this is not done the PIC16C710 will reset and wait for
    another 8 clocks. In this manner the master can reset the
    slave by toggling the clock once and waiting for 40 mS,
    which is the max. time for WDT with no postscaler.
```


© 1997 Microchip Technology Inc.

```
      OSC = 16Mhz; XT mode; WDT disabled; Microcontroller Mode;
                                                              */

                                #include <c:\mplabc\17C44.h>
                                #ifndef 17C44_H
                                /*
                           PIC17C44 Standard Header File, Version 1.00
                           (c) Copyright 1996 Microchip Technology, Inc., Byte Craft Limited

                                RAM locations reserved for temporary variables: 0x1A - 0x1E
                                */

                                #pragma option +1;
                                #endif

                                #include <c:\mplabc\delay16.h>
                                #pragma option +1;

00C7                            #define AdconDefault 0xC7 //
0000                            #define CLK      0        // PORTC bit 0 is the CLK
0001                            #define DO       1        // bit 1 is Data Out
0002                            #define DI       2        // bit 2 is data in
001F                            bits    ADCON;
0003                            #define Chs0     3
0004                            #define Chs1     4
0020                            char    Count;
0021                            char    Temp;
0022                            char    ADRES;
0023                            bits    Channel;        // tracks which channel is selected

0002                            #define SIOClkPulse   2     // CLK pulse in uS
0028                            #define MaxWdtTime    40    // WDT timeout in mS
0019                            #define AdAcqTime     25    // A/D Acquisition time in uS
0064                            #define DebounceTime  100   // Debounce time in mS
                                void InitADC(void)
                                {
0028 B800    MOVLB  00h             PORTB = 0;
0029 2912    CLRF   PORTB
002A B801    MOVLB  01h             PORTC = 0;
002B 2911    CLRF   PORTC
002C B800    MOVLB  00h             DDRB = 0;          // set PORTB as outputs
002D 2911    CLRF   DDRB
002E B0FC    MOVLW  FCh             DDRC = 0xFC;       // set PORTC for SPI
002F B801    MOVLB  01h
0030 0110    MOVWF  DDRC
0031 2923    CLRF   23              Channel = 0;
0032 B0C7    MOVLW  C7h             ADCON = AdconDefault;
0033 011F    MOVWF  1F
0034 0002    RETURN              }

                     /* The GetADC routine transmits the information on the channel to the
                        PIC16C710 while receiving the result of the last conversion.
                        The channel is selected by the value in Channel which goes from
                        0 to 3.                                                        */

                                char GetADC(void)
                                {
0035 B0C7    MOVLW  C7h             ADCON = AdconDefault;  // set ADCON to default
0036 011F    MOVWF  1F
0037 9023    BTFSS  23,0            if (Channel.0)         // select channel
0038 C03A    GOTO   003Ah
0039 831F    BSF    1F,3               ADCON.Chs0 = 1;
003A 9123    BTFSS  23,1            if (Channel.1)
003B C03D    GOTO   003Dh
003C 841F    BSF    1F,4               ADCON.Chs1 = 1;
```


© 1997 Microchip Technology Inc.

```
003D 2921    CLRF   21              Temp = 0;
003E 8804    BCF    ALUSTA,0        ALUSTA.C = 0;
003F 2920    CLRF   20              for(Count = 0;Count < 8;Count++)
0040 B008    MOVLW  08h
0041 0420    SUBWF  20,W
0042 9804    BTFSC  ALUSTA,0
0043 C05E    GOTO   005Eh
                                    {     // send 8 bit data
0044 191F    RRCF   1F              RRCF(ADCON);
0045 9004    BTFSS  ALUSTA,0        if (ALUSTA.C)
0046 C04A    GOTO   004Ah
0047 B8F1    MOVLB  F1h               PORTC.DO = 1;
0048 8111    BSF    PORTC,1
0049 C04C    GOTO   004Ch           else PORTC.DO = 0;
004A B8F1    MOVLB  F1h
004B 8911    BCF    PORTC,1
004C B8F1    MOVLB  F1h             PORTC.CLK = 1;
004D 8011    BSF    PORTC,0
004E B002    MOVLW  02h             Delay_Us_16MHz(SIOClkPulse);
004F E09C    CALL   009Ch
0050 B8F1    MOVLB  F1h             if (PORTC.DI)
0051 9211    BTFSS  PORTC,2
0052 C055    GOTO   0055h
0053 8004    BSF    ALUSTA,0          ALUSTA.C = 1;
0054 C056    GOTO   0056h           else ALUSTA.C = 0;
0055 8804    BCF    ALUSTA,0
0056 1921    RRCF   21              RRCF(Temp);
0057 B8F1    MOVLB  F1h             PORTC.CLK = 0;
0058 8811    BCF    PORTC,0
0059 B002    MOVLW  02h             Delay_Us_16MHz(SIOClkPulse);
005A E09C    CALL   009Ch
005B 8804    BCF    ALUSTA,0        ALUSTA.C = 0;
                                    }
005C 1520    INCF   20
005D C040    GOTO   0040h
005E 6A21    MOVFP  21,WREG         return(Temp);
005F 0002    RETURN
                                }

                /* The PIC16C710 has 4 channels each of which can be selected
                   by simply pressing the switch connected to RA1. This increments
                   the value of the Channel register from 0 to 3 and then back to 0.
                   The 100 mS delay is meant as a debounce delay.                  */

                                void SampleRA1Key(void)
                                {
0060 B800    MOVLB  00h             if(!PORTA.1)
0061 9910    BTFSC  PORTA,1
0062 C069    GOTO   0069h
0063                                {
0063 B064    MOVLW  64h             Delay_Ms_16MHz(DebounceTime);
0064 E086    CALL   0086h
0065 B800    MOVLB  00h             if(!PORTA.1)
0066 9910    BTFSC  PORTA,1
0067 C069    GOTO   0069h
0068 1523    INCF   23                Channel++;
                                    }
0069 B003    MOVLW  03h             if (Channel > 3)
006A 0423    SUBWF  23,W
006B 9204    BTFSS  ALUSTA,2
006C 9004    BTFSS  ALUSTA,0
006D C06F    GOTO   006Fh
006E 2923    CLRF   23              Channel = 0;
006F 0002    RETURN                 }
```


© 1997 Microchip Technology Inc.

```
                /* The entire SIO transfer from the PIC17C4X and the PIC16C710
                   should be done in less time than the max. WDT timeout time. If
                   in any case this does not occur, the PIC16C710 will experience a
                   WDT reset.  This "feature" can also be used to asyn. reset the
                   slave PIC16C710 as shown below.                                 */

                                    void ResetADC(void)
                                    {
0070 B801    MOVLB  01h                 PORTC.CLK = 1;
0071 8011    BSF    PORTC,0
0072 B002    MOVLW  02h                 Delay_Us_16MHz(SIOClkPulse);
0073 E09C    CALL   009Ch
0074 B801    MOVLB  01h                 PORTC.CLK = 0;
0075 8811    BCF    PORTC,0
0076 B028    MOVLW  28h                 Delay_Ms_16MHz(MaxWdtTime);
0077 E086    CALL   0086h
0078 0002    RETURN                     }

                /* On startup, the PIC17C4X resets the PIC16C710 by using the WDT
                   feature mentioned above.  It then requests an A/D value.
                   The A/D value of the channel selected by the "Channel" register.
                   It is then sampled by the PIC16C710 and the result is sent back
                   to the PIC17C4X and saved in ADRES.  The 8-bit value is displayed
                   on 8 LEDs connected to PORTB.  To select another channel, simply
                   press the switch connected to RA1 till the correct channel is
                   selected.                                                       */

                                    void main(void)
                                    {

0079 E028    CALL   0028h               InitADC();
007A E070    CALL   0070h               ResetADC();
007B E035    CALL   0035h               Temp = GetADC();
007C 0121    MOVWF  21
                                        while(1)
                                           {
007D B019    MOVLW  19h
007E E09C    CALL   009Ch
007F                                       Delay_Us_16MHz(AdAcqTime);
007F E035    CALL   0035h                  ADRES = GetADC();
0080 0122    MOVWF  22
0081 B800    MOVLB  00h                    PORTB = ADRES;
0082 7222    MOVFP  22,PORTB
0083 E060    CALL   0060h                  SampleRA1Key();
0084 C07D    GOTO   007Dh                  }
0085 0002    RETURN                     }
                                    #pragma option +1;
                                    /*
/*************************************************************************/
                                    void Delay_Ms_16MHz(registerw delay)
                                    /*
                                       Clock Speed = 16MHz
                                       Inst. Clock =  4MHz
                                       Inst. dur.  = 0.25us  */
0000                                {
                                    #asm
0086 0000                          NOP
0087 011A                          MOVWF   __WImage
                            DLMS16M1
                                   RADIX   DEC
0088 B0F9                          MOVLW   249
0089 0101                          MOVWF   FSR0
                            DLMS16M2
                            DLMS16M__  REPT  13
                                   NOP
                            ENDM
```


© 1997 Microchip Technology Inc.

```
008A 0000
008B 0000
008C 0000
008D 0000
008E 0000
008F 0000
0090 0000
0091 0000
0092 0000
0093 0000
0094 0000
0095 0000
0096 0000
0097 0117                   DECFSZ  FSR0
0098 8AC0                   GOTO    DLMS16M2

0099 1A17                   DECFSZ  __WImage
009A 88C0                   GOTO    DLMS16M1
                              #endasm
009B 0002    RETURN                 }
/*****************************************************************************/
                                    void Delay_Us_16MHz(registerw delay)
                                    /*
                                       Clock Freq. = 16MHz
                                       Inst. Clock = 4MHz
                                       Inst. dur.  = 250ns  */
0000                                {
                                    #asm
009C 0000                         NOP
009D 011A                         MOVWF   __WImage
                            DLUS16M
009E 0000                         NOP
009F 171A                         DECFSZ  __WImage
00A0 C09E                         GOTO    DLUS16M
                                    #endasm
00A1 0002    RETURN                 }
/*****************************************************************************/
ROM USAGE MAP

    0000 to 0000    0028 to 00A1
    Total ROM used 007B


Errors              :    0
Warnings            :    0
```


© 1997 Microchip Technology Inc.

APPENDIX B: CODE FOR PIC16C710 (SLAVE)

```
/* This program creates a serial I/O interface with a PIC17C4X
   device.  The basic intent of this interface is to use the
   PIC16C710 as a simple serial ADC device for use with the PIC17C4X.
   The 3 wire interface is as follows:
    CLK --> RB0/INT
    DI  --> RB1
    DO  --> RB2
   The PIC17C4X device will transmit an 8-bit word to the PIC16C710
   in order for it to configure the following:
    Bit 0 --> PCFG0
    Bit 1 --> PCFG1
    BIT 2 --> Start
    BIT 3 --> CHS0
    BIT 4 --> CHS1
    BIT 5 --> not used
    BIT 6 --> ADCS0
    BIT 7 --> ADCS1
   The ADCON0 and ADCON1 registers are merged into one register
   to help reduce the interface overhead. After all the 8 bits are
   received, bits 0 and 1 are placed in ADCON1 and bits 3 to 7
   are placed in ADCON0.  Depending on the state of the bit 2,
   the A/D conversion is then started.

   The protocol basics are as follows:
    Two registers are maintained in the PIC16C710 which are
    used in the transfer:
    ADCON --> ADCON1 merged with ADCON0
    ADRES --> same as the AD result register
    8 clock signals are used to transmit the ADRES value
    and receive the next ADCON value. On the rising edge of
    the clock the serial ADCON value is read whilst on the
    falling edge of the same clock the serial ADRES value
    is transmitted.  In this manner an efficient transfer
    rate is maintained with very little overhead to software
    and more importantly rate of A/D conversion.
    The PIC17C4X also maintains a replica of ADCON and ADRES
    and does just the opposite of what the PIC16C710 does i.e
    ADRES is received and ADCON is transmitted.  The clock
    master is always the PIC17C4X, the PIC16C710 is always the
    slave.  The transfer occurs as below:
    Clock   Data IN(DI) Data OUT(DO)
    ------------------------------------------------
    Falling edge   N/A     RRF ADRES; C --> DO
    Rising edge    read DI       N/A

    This is repeated till all 8 bits are transmitted/received.

    On receiving the 8-bit value, the ADCON1 and ADCON0 registers
    configured and the A/D conversion started.  The result of the
    conversion is then stored in the ADRES register.  The next
    transfer then sends the ADRES result out.

    NOTE: The PIC17C4X has to provide for enough time for sampling
    and for the conversion to complete.

   The WDT is enabled and used in the following manner:
    The entire 8 clocks and A/D conversion should be done in a
    max. of 8 mS, which is the min. timeout period for the WDT.
    If this is not done the PIC16C710 will reset and wait for
    another 8 clocks. In this manner the master can reset the
    slave by toggling the clock once and waiting for 40 mS,
    which is the max. time for WDT.
```


© 1997 Microchip Technology Inc.

```
        OSC = 20Mhz; HS mode; WDT enabled; PWRT timer disabled;
        */

                                #include <c:\mplabc\16c710.h>
                                #ifndef 16C710_H
                                /*
                        PIC16C710 Standard Header File, Version 1.00
                        (c) Copyright 1996 Microchip Technology, Inc., Byte Craft Limited

                        RAM locations reserved for temporary variables: 0x0C - 0x10
        */

                                #pragma option +l;

                                #endif

                                #include <c:\mplabc\delay14.h>
                                #pragma option +l;

0000                            #define CLK 0
0001                            #define DI  1
0002                            #define DO  2
0004                            #define Received 4
0006                            #define Complete 6
0011                            bits    ADCON;
0012                            char    Count;
0013                            bits    Flag;
0000                            #define NewCommand  0

0004                            #pragma vector Interrupt @ 0x0004;

         /* The whole process is driven by the completion of the interrupt routine
            and once the 1st clock is received, the interrupts are disabled. Because of this
            there is no reason to save the W and STATUS registers during interrupts */

0004 2805    GOTO   0005h       void Interrupt(void)
0005                            {
0005 1283    BCF    03,5            PORTB.Complete = 0;
0006 1306    BCF    06,6
0007 108B    BCF    0B,1            INTCON.INTF = 0;     // clr flag
                                    do          // first int is on rising edge so read data
                                       {
0008 1283    BCF    03,5               while(!PORTB.CLK); //make sure rising edge is present
0009 1C06    BTFSS  06,0
000A 2808    GOTO   0008h
000B 1283    BCF    03,5               if (PORTB.DI)        // Read Data in
000C 1C86    BTFSS  06,1
000D 2810    GOTO   0010h
000E 1403    BSF    03,0                 STATUS.C = 1;
000F 2811    GOTO   0011h               else STATUS.C = 0;
0010 1003    BCF    03,0
0011 0C91    RRF    11                 RRCF(ADCON);   // get data in ADCON
0012 0C89    RRF    09                 RRCF(ADRES);   // prepare data out from ADRES
0013 1C03    BTFSS  03,0               if (STATUS.C)
0014 2817    GOTO   0017h
0015 1506    BSF    06,2                 PORTB.DO = 1;    //set data out
0016 2818    GOTO   0018h               else PORTB.DO = 0;
0017 1106    BCF    06,2
0018 1283    BCF    03,5               while(PORTB.CLK); // wait for falling edge
0019 1806    BTFSC  06,0
001A 2818    GOTO   0018h
001B 1283    BCF    03,5               Count++;   // increment count
001C 0A92    INCF   12
001D 3008    MOVLW  08h                }while(Count < 8);
001E 0212    SUBWF  12,W
001F 1C03    BTFSS  03,0
```


© 1997 Microchip Technology Inc.

```
0020 2808    GOTO   0008h
0021 1283    BCF    03,5                Flag.NewCommand = 1;       //set flag
0022 1413    BSF    13,0
0023 120B    BCF    0B,4                INTCON.INTE = 0;
0024 1606    BSF    06,4                PORTB.Received = 1;        // set "Received" LED
0025 0009    RETFIE                   }

                                      void Convert(void)
                                      {
0026 1283    BCF    03,5                Flag.NewCommand = 0;
0027 1013    BCF    13,0
0028 0192    CLRF   12                  Count = 0;
0029 3003    MOVLW  03h                 ADCON1 = ADCON & 0x03;
002A 0511    ANDWF  11,W
002B 1683    BSF    03,5
002C 0088    MOVWF  08
002D 30F8    MOVLW  F8h                 ADCON0 = ADCON & 0xF8;
002E 1283    BCF    03,5
002F 0511    ANDWF  11,W
0030 0088    MOVWF  08
0031 1408    BSF    08,0                ADCON0.ADON = 1;
0032 1D11    BTFSS  11,2                if(ADCON.GO)
0033 2838    GOTO   0038h
0034                                      {
0034 1508    BSF    08,2                ADCON0.GO = 1;
0035 1283    BCF    03,5                while (ADCON0.ADIF == 1); // wait till conv. is done
0036 1888    BTFSC  08,1
0037 2835    GOTO   0035h
                                       }
0038 160B    BSF    0B,4                INTCON.INTE = 1;             // enable interrupts on edge
0039 1283    BCF    03,5                PORTB.Complete = 1;          // set completion LED
003A 1706    BSF    06,6
003B 1206    BCF    06,4                PORTB.Received = 0;
003C 0008    RETURN                   }

                                      void main(void)
                                      {
003D 1283    BCF    03,5                PORTB = 0x00;
003E 0186    CLRF   06
003F 3003    MOVLW  03h                 TRISB = 0x03;
0040 1683    BSF    03,5
0041 0086    MOVWF  06
0042 1283    BCF    03,5                Count = 0;
0043 0192    CLRF   12
0044 30F8    MOVLW  F8h                 OPTION = OPTION & 0xF8;        // WDT = 18 mS
0045 1683    BSF    03,5
0046 0581    ANDWF  01
0047 1701    BSF    01,6                OPTION.INTEDG = 1;             // int on rising edge
0048 018B    CLRF   0B                  INTCON = 0;
0049 160B    BSF    0B,4                INTCON.INTE = 1;               // enable interrupt
004A 178B    BSF    0B,7                INTCON.GIE = 1;
004B 1283    BCF    03,5                Flag.NewCommand = 0;
004C 1013    BCF    13,0
                                       while (1)
                                          {
004D 1283    BCF    03,5                if(Flag.NewCommand)
004E 1813    BTFSC  13,0
004F 2026    CALL   0026h                 Convert();
0050 0064    CLRWDT                     CLRWDT();
0051 284D    GOTO   004Dh                }
0052 0008    RETURN                   }

0000 3000    MOVLW  00h
0001 008A    MOVWF  0A
0002 283D    GOTO   003Dh
```

9

```
ROM USAGE MAP

    0000 to 0002    0004 to 0052
    Total ROM used 0052


Errors              :    0
Warnings            :    0
```


© 1997 Microchip Technology Inc.

Using SRAM With A PIC16CXXX

Author: Rick Evans
Microchip Technology Inc.

INTRODUCTION

There are applications where a significant amount of data memory is required beyond what is in the microcontroller. For example, buffering communications data, creating large volatile tables and arrays. One interesting application is voice storage and playback. Some applications require relatively low frequencies, so a fast address/data bus and expensive FSRAM are not necessary.

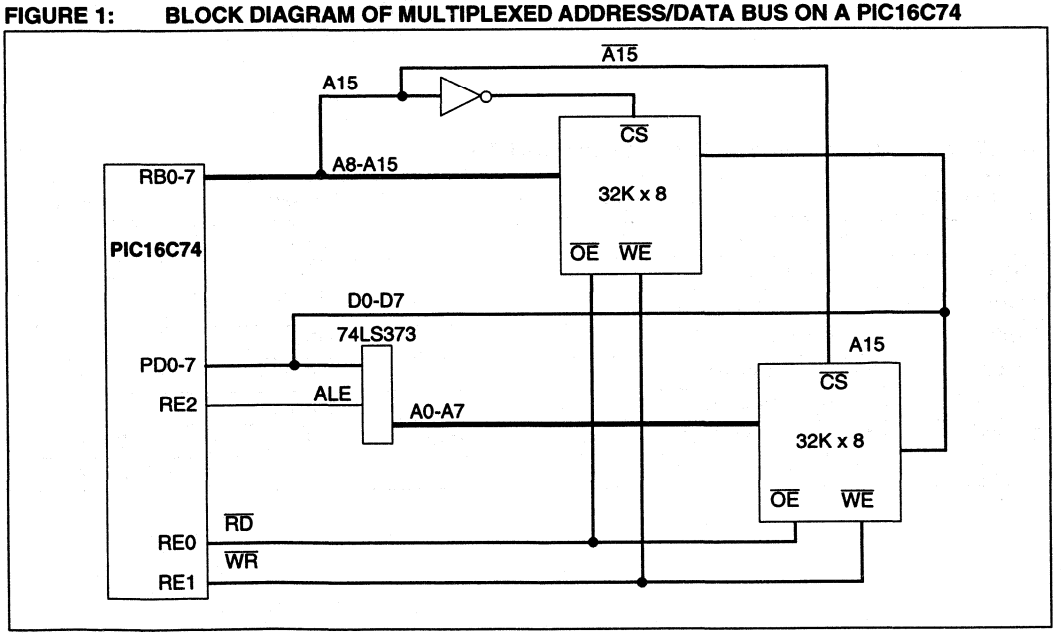
This application note uses the PIC16C74. Since the PIC16C74 does not have an external address/data bus, one was created using the I/O ports. A software implementation of a multiplexed address/data bus is more than adequate for some applications. This application note discusses the read and write speeds achievable with the PIC16C74 running at 20 MHz, hardware connections necessary, and software routines for reading and writing to external memory.

IMPLEMENTATION

The multiplexed address/data bus was created using nineteen port pins (Figure 1). PORTD is address lines 0 through 7 multiplexed with data lines 0 through 7. PORTB is the upper address lines 8-15. PORTE<0> is the $\overline{RD}$, PORTE<1> is $\overline{WR}$ and PORTE<2> is ALE (address latch enable). One SRAM chip is address range 0000h-7FFFh, and the other SRAM chip is address range 8000h-FFFFh. The chip selecting is done using A15 and $\overline{A15}$. A 74LS373 latch is needed to demultiplex the low order address/data bus.

The software needed to initialize the ports is in the subroutine `init_muxbus` (Appendix A). The initial states of the address lines and bus control signals are shown in the comments of the subroutine.

FIGURE 1: BLOCK DIAGRAM OF MULTIPLEXED ADDRESS/DATA BUS ON A PIC16C74



9

© 1997 Microchip Technology Inc.

READ CYCLE

The software to do the read cycle is very straight forward. The read cycle drives the address onto the 16-bit bus, then latches the low order address lines 0-7 on to the memory device (Figure 2). When ALE goes low, address 0-7 is latched. Then the low order bus lines 0-7 are changed to input pins. Next, the read pin ($\overline{RD}$) goes low which turns on the output buffers of the memory device. Some time later, the data from the memory is driven on to the data lines 0-7. Then $\overline{RD}$ goes high and the output buffers of the memory device are disabled. Next, the low order data lines are changed back to outputs. The subroutine `read_extmem` shows the code used to emulate a read cycle on PORTB and PORTD.

You can ascertain from Table 1, that a slow SRAM can be used. There are three critical SRAM read cycle specifications:

TACC: Address access time
TOE: Output enable time
TDF: Data float time

The address access time TACC corresponds to the TAVDV (1.6 µs) of the emulated muxed bus. A FUJITSU MB84256C-70 has an access time of 70 ns. The output enable time TOE from the Fujitsu data sheet is 35 ns. The emulated muxed bus specification TRLDV (200 ns) corresponds to the TOE of the Fujitsu SRAM. The data float time TDF is only important if you are doing back to back bus cycles.

WRITE CYCLE

The write cycle moves data to the external SRAM. The address is driven on the sixteen address lines, then the low order address is latched by making ALE go high and then low. The data is moved to the low order address lines (data lines). Finally, the $\overline{WR}$ pin is driven low and then high. When write goes high, the data is written to the SRAM. Figure 3 shows the bus timing diagram for the write cycle.

The write cycle is even simpler to implement in software than the read cycle. The subroutine `write_extmem` shows code used to write out data to an external SRAM.

The write pulse width is an important specification in memory design as well. For the PIC16C74 emulated address/data bus the write pulse width is TWLWH. The value for the write pulse width is very large compared to actual address/data bus implementations. The TAVWH specification corresponds to the TAW specification of most memory devices. The emulated address/data bus has relatively slow bus timings compared with most memory access times such as EPROMs and SRAMs.

SUMMARY

A 40-pin PIC16CXXX device such as the PIC16C74 can interface to external memory. This application note used sixteen address lines multiplexed with eight data lines to read/write to two external SRAMs. The read/write cycle times can be calculated easily given the single cycle instruction architectures of the PIC16CXXX. Flash, EPROM and other parallel bus memory devices can be similarly utilized.

TABLE 1: READ AND WRITE CYCLE BUS TIMINGS

Timing	Description	Minimum	Maximum
TCY	Instruction cycle time @ 20 MHz	200 ns	DC
TLLLH	ALE pulse width	1 TCY	—
TAVDV	Address valid to data valid	—	7 TCY
TRLDV	Read low to data valid	1 TCY	—
TRHDZ	Read high to data float	0	1 TCY
TWLWH	WRITE pulse width	1 TCY	—
TWHDX	WRITE high to data no longer valid (data hold time)	—	2 TCY
TAVWH	Address valid to write high	—	5 TCY

© 1997 Microchip Technology Inc.

FIGURE 2: READ CYCLE ON MULTIPLEXED ADDRESS/DATA BUS

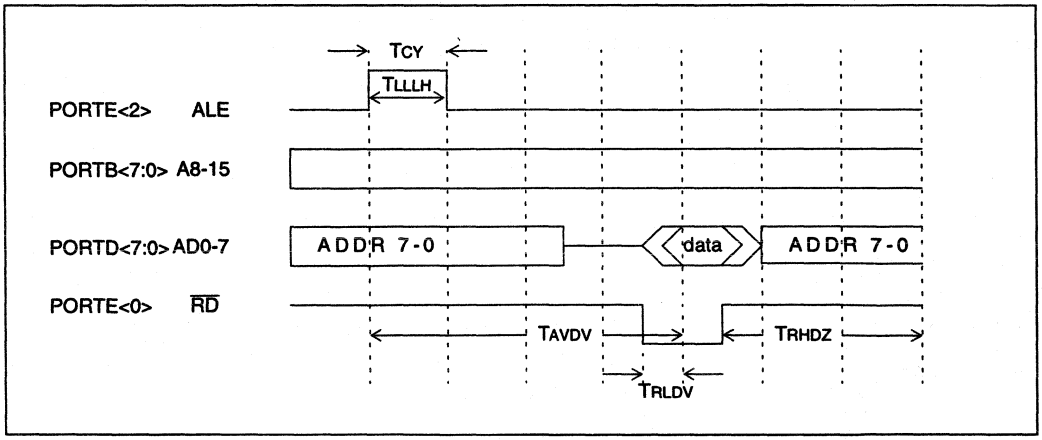
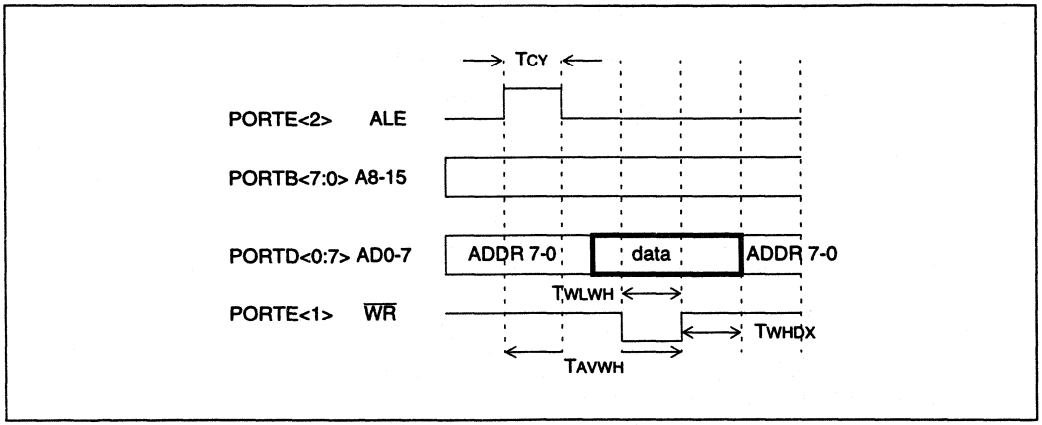


FIGURE 3: WRITE CYCLE BUS TIMINGS



9

APPENDIX A:PROGRAM LISTING: EXTERNAL MEMORY SUBROUTINES

```
;*****************************************************************
;*           Initialize the Multiplexed Address/Data Bus
;*
;* AD0-AD7 is PORTD 0-7
;* A8-A15 is PORTB 0-7
;* ALE is PORTE.2
;* RD# is PORTE.0
;* WR# is PORTE.1
;*
;* This init routine sets the multiplexed address/data bus up as
;* A0-A15 --> output low
;* ALE --> OUTPUT LOW
;* RD#,WR# --> OUTPUT HIGH
;*
;*****************************************************************

Init_MUXBUS
        bsf     STATUS,RP0          ;switch to bank 1 registers
        clrf    TRISB               ;set A8-A15 as output
        clrf    TRISD               ;set AD0-AD7 as output
        movlw   0xf8
        andwf   TRISE,F             ;ALE,RD#,WR# as output

        bcf     STATUS,RP0          ;switch to bank 0 registers

        clrf    ADHIGH              ;set A8-A15 to 0 (PORTB)
        clrf    ADLOW               ;set AD0-AD7 to 0 (PORTD)
        movlw   3                   ;ale=0,rd#=1,wr#=1
        movwf   PORTE
      return


;*****************************************************************
;*              Read External Memory of muxed bus
;*
;* INPUT: PORTB =A8-A15, PORTD = AD0-AD7
;* OUTPUT: W reg contains 8-bit data read from ext. mem.
;* CHANGED: W reg, ALE, RD#
;* (This READ routine has been modified to save the low order
;* address before a READ is done. The data read from memory will
;* destroy the address. After the read is done, the low order address
;* is written back out to PORTD.)
;*
;*****************************************************************
read_extmem
        movf    ADLOW,W             ;save low order address
        movwf   ADLOW_IMAGE

        bsf     PORTE,ALE           ;ALE high for 200ns, RD#, WR# low
        bcf     PORTE,ALE           ;ALE goes low (A0-7 latched)
        bsf     STATUS,RP0
        movlw   0xff
        movwf   TRISD               ;make PORTD input
        bcf     STATUS,RP0

        bcf     PORTE,RD            ;drop READ low

        movf    ADLOW,W             ;move read data from AD bus to w reg
        bsf     PORTE,RD            ;pull READ high (RD pulse is 400ns)
        bsf     STATUS,RP0
        clrf    TRISD               ;make PORTD (ADLOW) output again
        bcf     STATUS,RP0

        movwf   w_image            ;save READ data
        movf    ADLOW_IMAGE,W      ;restore low order address
```


© 1997 Microchip Technology Inc.

```
        movwf   ADLOW               ;on port
        movf    w_image,W           ;restore READ data to w

        return
;*********************************************************************
;*              Write to External Memory on muxed bus
;* INPUT:  PORTB= A8-A15, PORTD = AD0-AD7, W= 8-bit data to write
;* OUTPUT: NOTHING
;* CHANGED: PORTE IS TOGGLED FOR ALE,WR# AND PUT BACK TO 011B
;* (This WRITE routine has been modified to save the low order
;*  address before a write is done. Then the low order address
;* is put back on PORTD after the write.)
;*********************************************************************
write_extmem
        movwf   w_image             ;save w (data to write)
        movf    ADLOW,W
        movwf   ADLOW_IMAGE         ;save the low order address
        movf    w_image,W           ;restore w (data to write)

        bsf     PORTE,ALE           ;ALE high for 200ns, RD#,WR# low
        bcf     PORTE,ALE           ;latch lower address
        movwf   ADLOW               ;move write data to AD0-7
        bcf     PORTE,WR            ;WR# low for 200ns
        bsf     PORTE,WR            ;latch data in external memory

        movf    ADLOW_IMAGE,W
        movwf   ADLOW               ;restore low order address

        return
```


© 1997 Microchip Technology Inc.

NOTES:

© 1997 Microchip Technology Inc.

SECTION 10
REFERENCE DESIGNS

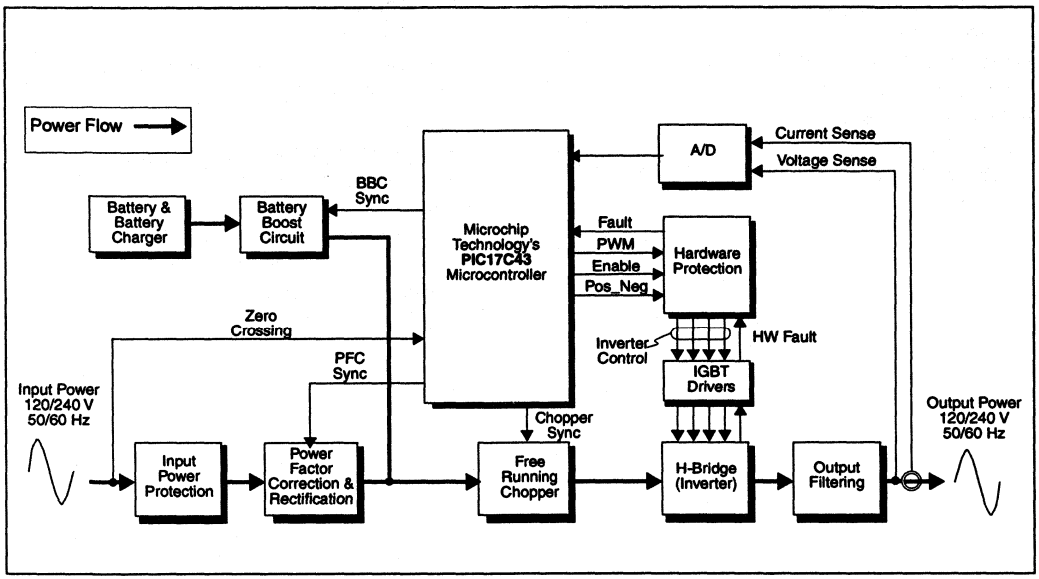
Uninterruptible Power Supply Reference Design Based on PIC17C43 - RD001 10-1
Intelligent Battery Charger Reference Design Based on PIC16C7X - RD002 10-3
Watt-Hour Meter Reference Design Based on PIC16C924 - RD003 10-5

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

RD001

Uninterruptible Power Supply Reference Design Based on PIC17C43



The PICREF-1 Reference Design offers a ready-made uninterruptible power supply solution with the flexibility of a microcontroller.

The Microchip Technology PICREF-1 Reference Design offers designers an intelligent UPS solution. This reference design is based on the "on-line" topology which provides true UPS performance.

The PIC17C43 microcontroller handles all the control of the UPS system. The PIC17C43 is unique because it provides a high performance and low cost solution not found in other microcontrollers.

The PIC17C43 PWM controls an inverter as necessary for generation of a sinusoidal AC output waveform. Fault signaling can be initiated internal or external to the PIC17C43 depending on the type of fault. A fault will disable the entire inverter. The output voltage and current will be monitored by the PIC17C43 to make adjustments "real-time" to correct for DC offset and load changes.

The PIC17C43 controls all module synchronization as well as inverter control and feedback. The PIC17C43 uses zero crossing for synchronization of input voltage/phase to output voltage/phase. All internal module synchronization is handled by the PIC17C43.

The control algorithms and software are written in C for maintainability and transportability.

To obtain a copy of the complete reference design documentation, contact your nearest Microchip Worldwide Sales and Service Representative and ask about the PICREF-1 Uninterruptible Power Supply Reference Design.

10

PICREF-1 FEATURES

- On-Line Topology
- True Sinusoidal Output
- 1400 VA Rating
- 120/240 V Input

© 1997 Microchip Technology Inc.

MICROCONTROLLER BENEFITS

- High Quality Sine Wave
  - Harvard Architecture
  - Hardware Multiply
  - 10-bit PWM
- Flexibility
- Transportability of Design
- Variable Loop Response
- Digital Filtering
- Parts and Complexity Reduction
- Peripheral Integration
- Ease of Interfacing
- Testability
- Time to Market

DOCUMENTATION YOU GET

- Theory and operation of UPS
- Fully documented source code
- Information on how to build a demo unit
  - Full circuit schematics
  - Bill of materials
  - PCB layout and Fab drawings
  - Unit assembly
  - Test results

Reference design material is available electronically from the following sources:

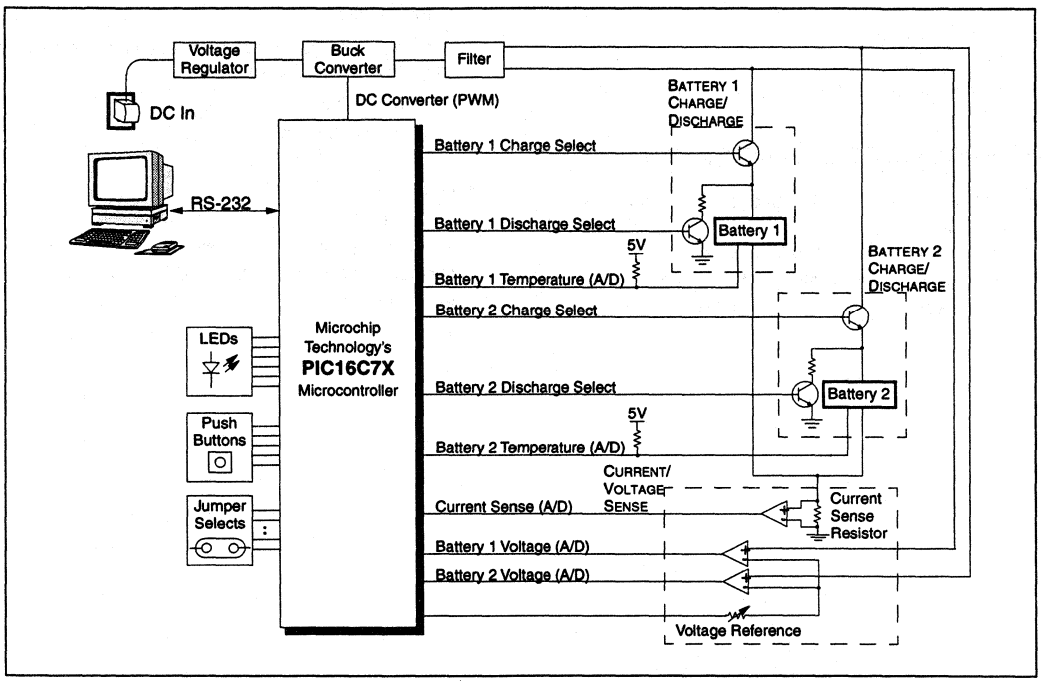
- Worldwide Web
  - www.microchip.com
  - ftp.mchip.com/biz/mchip
- BBS
  - from internet: mchipbbs.microchip.com
  - from compuserve: MCHIPBBS
- Microchip Technical Library on CD-ROM (future release)

© 1997 Microchip Technology Inc.

MICROCHIP

RD002

Intelligent Battery Charger Reference Design Based on PIC16C7X



The PICREF-2 Reference Design offers a ready-made battery charger solution that can accommodate several popular types of batteries.

With the PICREF-2 Reference Design, the user will be able to simply pick their complete battery charging system by completing the steps listed below:

1. Pick the required battery management features from the modular source code provided.
2. Pick the critical battery pack parameters and modify the global constants to those specifications.

The hardware design contains the necessary circuitry to support charging and discharging algorithms for four battery technologies (NiCd, NiMH, LiI, and Pb-Acid), four charge termination methods (-ΔV, ΔT/Δt, Absolute Voltage, and Timed Charge) and RS-232 communications.

The modular source code is written in C and consists of the four charge termination algorithms (-ΔV, ΔT/Δt, Absolute Voltage, and Timed Charge), discharge algorithm, inter-device communications and RS-232 communications modules.

The PC-based software provides a means for requesting and displaying battery status information.

To obtain a copy of the complete reference design, contact your nearest Microchip Worldwide Sales and Service Representative and ask for the PICREF-2 Intelligent Battery Charger Reference Design.

10

© 1997 Microchip Technology Inc.

PICREF-2 FEATURES

- Compatibility Across Technologies - (NiCd, NiMH, Li-Ion and Lead Acid)
- Low Cost
- Flexible Development Environment
- Fast Charge Rate
- High Charge Current Capability
- High Discharge Current Capability for Conditioning
- Real-Time Debug
- Data Logging
- User Selectable Embedded Charger Termination Algorithm

MICROCONTROLLER BENEFITS

- Flexibility to handle different chemistries, voltages and capacities.
- Variable Voltage Generation Control
- Charge/Discharge Multiple Battery Packs
- "Windowed" A/D for High Resolution

DOCUMENTATION YOU GET

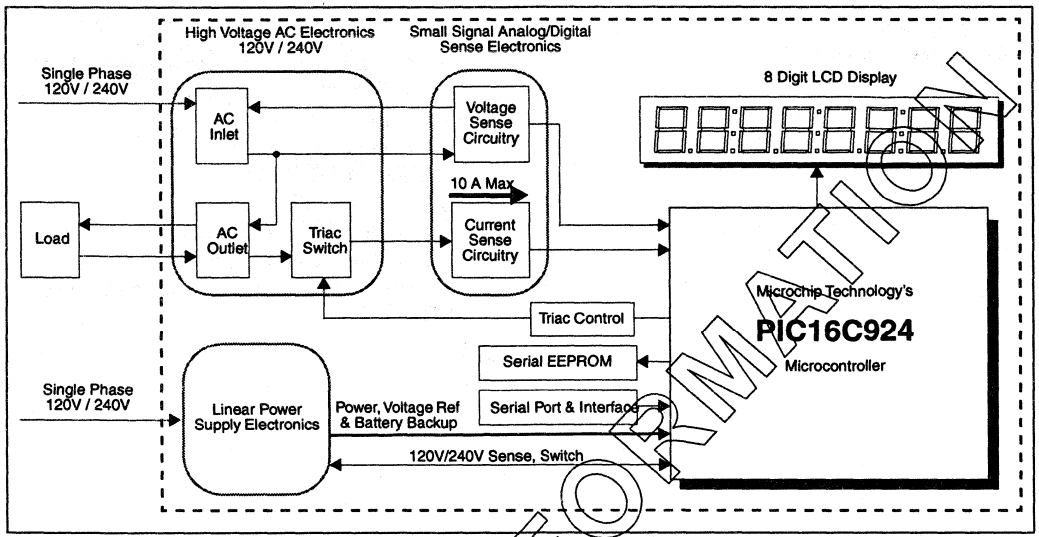
- Theory and operation of Battery Charger
- Fully documented source code
- Information on the demo unit
  - Full circuit schematics
  - Bill of materials
  - PCB layout and Fab drawings
  - Unit assembly
  - Test results

Reference design material is available electronically from the following sources:

- Worldwide Web
  - www.microchip.com
  - ftp.mchip.com/biz/mchip
- BBS
  - from internet: mchipbbs.microchip.com
  - from compuserve: MCHIPBBS
- Microchip Technical Library on CD-ROM (future release)

© 1997 Microchip Technology Inc.

Watt-Hour Meter Reference Design Based on PIC16C924



The PICREF-3 Reference Design provides a cost-effective way to monitor and display power and energy consumption for a single 120V or 240V electrical service.

The PICREF-3 Reference Design shows how to develop a cost-effective Watt-Hour meter. Using the PIC16C924 microcontroller, the meter can interleave real-time voltage and current waveform sampling with power and energy calculations.

Single phase (120V or 240V) electricity is sensed by the high voltage AC electronics. Balanced input voltage and current signals are then provided to the Small Signal Analog/Digital Sense Electronics for input into the PIC16C924. The PIC16C924 uses the analog/digital voltage sense and analog/digital current sense to calculate the power and energy consumption of the load. The microcontroller then displays this on the 8-digit LCD. Linear Power Supply Electronics provide general device power, voltage reference, battery backup and 120V/240V sense information.

The modular source code is written in C. Software algorithms accomplish voltage and current measurements, voltage and current waveform phase shift measurements, real-time timekeeping, and calculation and display of energy consumption. In addition, there is code to display measurements on the LCD and to communicate data to the serial EEPROM or via the serial port.

To obtain a copy of the complete reference design, contact your nearest Microchip Worldwide Sales and Service Representative and ask for the PICREF-3 Watt-Hour Meter Reference Design.

PICREF-3 FEATURES

- Low Cost
- 120V/240V Automatic Switch
- Flexible Development Environment
- Battery Backup

ADVANCE INFORMATION

MICROCONTROLLER BENEFITS

- Real-Time Electrical Measurement and Power/ Energy Calculations
- Direct LCD Drive
  - Present Time and Date
  - Total Whr
  - Alternate Energy (VARhr or VAhr)
  - Maximum or Cumulative Demand
- Customization
- Quick Time to Market

DOCUMENTATION YOU GET

- Theory and operation of Watt-Hour Meter
- Fully documented source code
- Information on the demo unit
  - Full circuit schematics
  - Bill of materials
  - PCB layout and Fab drawings
  - Unit assembly instructions
  - Test results

Reference design material is available electronically from the following sources:

- Worldwide Web
  - www.microchip.com
  - ftp.mchip.com/biz/mchip
- BBS
  - from internet: mchipbbs.microchip.com
  - from compuserve: MCHIPBBS
- Microchip Technical Library on CD-ROM (future release)

ADVANCE INFORMATION

 © 1997 Microchip Technology Inc.

SECTION 11
DEVELOPMENT TOOLS

System Support	Development Tools Selection Chart	11-1
On-Line Support	Microchip Internet and BBS Connections	11-1
MPLAB™	Integrated Development Environment	11-9
MPASM	Universal PIC16/17 Microcontroller Assembler Software	11-11
MPLAB™SIM	Software Simulator	11-13
MPLAB™C	Universal C Compiler for PICmicro MCUs	11-15
MP-Driveway™	Application Code Generator	11-17
***fuzzy*TECH®MP**	Fuzzy Logic Development System for PICmicro MCUs	11-19
PICMASTER®CE	CE Compliant In-Circuit Emulator	11-21
PICMASTER®	Full-Featured Modular In-Circuit Emulator	11-23
ICEPIC™	Low-Cost PIC16CXXX In-Circuit Emulator	11-25
PRO MATE®II	Universal Device Programmer	11-27
PICSTART®Plus	Low Cost Development Kit	11-29
PICSTART®Lite	Ultra Low Cost Development Kit	11-31
KEELOQ®	HCSXXX Device Programmers	11-33
KEELOQ®	HCSXXX Evaluation Kit	11-35
PICDEM-1	Low-Cost PIC16/17 Demonstration Board	11-37
PICDEM-2	Low-Cost PIC16CXX Demonstration Board	11-39
PICDEM-3	Low-Cost PIC16C9XX Demonstration Board	11-41
Memory Product Tools:		
	Microchip Serial EEPROM Designer's Kit	11-43
	Microchip Serial EEPROM Endurance Model	11-45

11

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

Development Tools Selection Charts

Emulator Systems Cross Reference and Ordering Information

Silicon Device	Model Name							
	PICMASTER with PRO MATE II	PICMASTER without PRO MATE II	PICMASTER-CE with PRO MATE II	PICMASTER-CE without PRO MATE	PICMASTER Probe Kit	PICMASTER-CE Probe Kit	ICEPIC System	ICEPIC Daughter Board
Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #
PIC12C508*	EM167015	EM167016	EM167101	EM167102	AC165004	AC165015	EM167201	AC165201
PIC12C509*	EM167015	EM167016	EM167101	EM167102	AC165004	AC165015	EM167201	AC165201
PIC12C671*	Planned	Planned	Planned	Planned	Planned	Planned	Planned	Planned
PIC12C672*	Planned	Planned	Planned	Planned	Planned	Planned	Planned	Planned
PIC14C000	EM147001	EM147002	EM147101	EM147102	AC145001	AC145002	Planned	Planned
PIC16C52	EM167015	EM167016	EM167101	EM167102	AC165004	AC165015	EM167201	AC165201
PIC16C54	EM167015	EM167016	EM167101	EM167102	AC165004	AC165015	EM167201	AC165201
PIC16C54A	EM167015	EM167016	EM167101	EM167102	AC165004	AC165015	EM167201	AC165201
PIC16C55	EM167015	EM167016	EM167101	EM167102	AC165004	AC165015	EM167201	AC165201
PIC16C56	EM167015	EM167016	EM167101	EM167102	AC165004	AC165015	EM167201	AC165201
PIC16C57	EM167015	EM167016	EM167101	EM167102	AC165004	AC165015	EM167201	AC165201
PIC16C58A	EM167015	EM167016	EM167101	EM167102	AC165004	AC165015	EM167201	AC165201
PIC16C554	EM167033	EM167034	EM167113	EM167114	AC165030	AC165020	EM167208	AC165208
PIC16C558	EM167033	EM167034	EM167113	EM167114	AC165030	AC165020	EM167208	AC165208
PIC16C61	EM167021	EM167022	†	†	AC165007	AC165017	EM167205	AC165205
PIC16C62A	EM167025	EM167026	EM167103	EM167104	AC165009	AC165016	EM167207	AC165207
PIC16C63	EM167025	EM167026	EM167103	EM167104	AC165009	AC165016	EM167207	AC165207
PIC16C64A	EM167025	EM167026	EM167103	EM167104	AC165009	AC165016	EM167207	AC165207
PIC16C65A	EM167025	EM167026	EM167103	EM167104	AC165009	AC165016	EM167207	AC165207
PIC16C620	EM167023	EM167024	EM167109	EM167110	AC165008	AC165018	EM167202	AC165202
PIC16C621	EM167023	EM167024	EM167109	EM167110	AC165008	AC165018	EM167202	AC165202
PIC16C622	EM167023	EM167024	EM167109	EM167110	AC165008	AC165018	EM167202	AC165202
PIC16C642	EM167035	EM167036	EM167115	EM167116	AC165031	AC165021	Planned	Planned
PIC16C662	EM167035	EM167036	EM167115	EM167116	AC165031	AC165021	Planned	Planned
PIC16C71	EM167027	EM167028	EM167105	EM167106	AC165010	AC165013	EM167205	AC167205
PIC16C710	EM167027	EM167028	EM167105	EM167106	AC165010	AC165013	Planned	Planned
PIC16C711	EM167027	EM167028	EM167105	EM167106	AC165010	AC165013	Planned	Planned
PIC16C715	EM167037	EM167038	EM167117	EM167118	AC165032	AC165022	Planned	Planned
PIC16C72	EM167025	EM167026	EM167103	EM167104	AC165009	AC165016	EM167207	AC165207
PIC16C73A	EM167025	EM167026	EM167103	EM167104	AC165009	AC165016	EM167207	AC165207
PIC16C74A	EM167025	EM167026	EM167103	EM167104	AC165009	AC165016	EM167207	AC165207
PIC16F83	EM167029	EM167030	EM167107	EM167108	AC165011	AC165014	Planned	Planned
PIC16F84	EM167029	EM167030	EM167107	EM167108	AC165011	AC165014	Planned	Planned
PIC16C923	EM167031	EM167032	EM167111	EM167112	AC165012	AC165019	EM167210	AC165210

Note: * PICMASTER PIC12CXXX emulation support also requires the use of a probe kit daughter board AC122001

* ICEPIC PIC12CXXX emulation support also requires the use of a daughter board adapter AC122002

**Contact Factory for availability

Shaded area indicates not applicable

† Recommend using PIC16C71 emulator systems (EM167105 or EM167106) or use AC165017 probe kit with existing emulator system.

© 1997 Microchip Technology Inc.

Emulator Systems Cross Reference and Ordering Information (Continued)

Silicon Device	Model Name							
	PICMASTER with PRO MATE II	PICMASTER without PRO MATE II	PICMASTER-CE with PRO MATE II	PICMASTER-CE without PRO MATE	PICMASTER Probe Kit	PICMASTER-CE Probe Kit	ICEPIC System	ICEPIC Daughter Board
Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #
PIC16C924	EM167031	EM167032	EM167111	EM167112	AC165012	AC165019	EM167210	AC165210
PIC17C42A	EM177007	EM177008	EM177107	EM177108	AC175002	AC175003		
PIC17C43	EM177007	EM177008	EM177107	EM177108	AC175002	AC175003		
PIC17C44	EM177007	EM177008	EM177107	EM177108	AC175002	AC175003		
PIC17C756	EM177009**	EM177010**	EM177109**	EM177110**	AC175004**	AC175005**		

Note: * PICMASTER PIC12CXXX emulation support also requires the use of a probe kit daughter board AC122001

* ICEPIC PIC12CXXX emulation support also requires the use of a daughter board adapter AC122002

**Contact Factory for availability

Shaded area indicates not applicable

† Recommend using PIC16C71 emulator systems (EM167105 or EM167106) or use AC165017 probe kit with existing emulator system.

Software Tools

Silicon Device	Model Name					
	MPLAB	MPLAB-C	*fuzzy*TECH-MP Explorer	*fuzzy*TECH-MP Edition	MP-DriveWay	Total Endurance Software Model
Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #
24CXX/24LCXX						SW242001
25CXX/25LCXX						SW242001
93CXX/93LCXX						SW242001
59C11						SW242001
PIC12C508	SW007002	SW006005	DV005001	DV005002	Planned	
PIC12C509	SW007002	SW006005	DV005001	DV005002	Planned	
PIC12C671	SW007002	SW006005	DV005001	DV005002	Planned	
PIC12C672	SW007002	SW006005	DV005001	DV005002	Planned	
PIC14C000	SW007002	SW006005	DV005001	DV005002	Planned	
PIC16C52	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C54	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C54A	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C55	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C56	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C57	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C58A	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C554	SW007002	SW006005	DV005001	DV005002	Planned	
PIC16C558	SW007002	SW006005	DV005001	DV005002	Planned	
PIC16C61	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C62A	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C63	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C64A	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C65A	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C620	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C621	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C622	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C642	Planned	Planned	DV005001	DV005002	Planned	
PIC16C662	Planned	Planned	DV005001	DV005002	Planned	

Shaded area indicates not applicable.

**Contact Factory for availability

© 1997 Microchip Technology Inc.

Software Tools (Continued)

Silicon Device	Model Name					
	MPLAB	MPLAB-C	*fuzzy*TECH-MP Explorer	*fuzzy*TECH-MP Edition	MP-DriveWay	Total Endurance Software Model
Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #
PIC16C71	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C710	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C711	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C715	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C72	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C73A	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C74A	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C76	SW007002	SW006005	DV005001	DV005002	Planned	
PIC16C77	SW007002	SW006005	DV005001	DV005002	Planned	
PIC16F83	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16F84	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC16C923	SW007002	SW006005	DV005001	DV005002	Planned	
PIC16C924	SW007002	SW006005	DV005001	DV005002	Planned	
PIC17C42A	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC17C43	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC17C44	SW007002	SW006005	DV005001	DV005002	SW006006	
PIC17C756	SW007002	SW006005**	DV005001	DV005002	SW006006	

Shaded area indicates not applicable.
**Contact Factory for availability

Programmers Cross Reference and Ordering Information

Silicon Device	Model Name									
	PICSTART Lite	PICSTART Plus	KEELOQ Programmer	PRO MATE II	PRO MATE II Socket Modules					
					DIP	SOIC	SSOP	PLCC	MQFP	TQFP
Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #
24CXX				DV007003	AC004001					
25CXX				DV007003	AC004001					
93CXX				DV007003	AC004001					
HCS200			PG306001	DV007003	AC004001	AC004002				
HCS300			PG306001	DV007003	AC004001	AC004002				
HCS301			PG306001	DV007003	AC004001	AC004002				
HCS360			PG306001	DV007003	AC004001	AC004002				
HCS361			PG306001	DV007003	AC004001	AC004002				
HCS509			PG306001	DV007003	AC004001	AC004002				
HCS512			PG306001	DV007003	AC004001	AC004002				
PIC12C508		DV003001		DV007003	AC124001	AC124001				
PIC12C509		DV003001		DV007003	AC124001	AC124001				
PIC12C671		DV003001		DV007003	AC124001	AC124001				
PIC12C672		DV003001		DV007003	AC124001	AC124001				
PIC14C000		DV003001		DV007003	AC144001	AC144002	AC144002			
PIC16C52	DV162003	DV003001		DV007003	AC164001	AC164002	AC164015			
PIC16C54	DV162003	DV003001		DV007003	AC164001	AC164002	AC164015			

Shaded area indicates not applicable.
**Contact Factory for Availability

11

Programmers Cross Reference and Ordering Information (Continued)

Silicon Device	Model Name									
	PICSTART Lite	PICSTART Plus	KEELOQ Programmer	PRO MATE II	PRO MATE II Socket Modules					
					DIP	SOIC	SSOP	PLCC	MQFP	TQFP
Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #
PIC16C54A	DV162003	DV003001		DV007003	AC164001	AC164002	AC164015			
PIC16C55	DV162003	DV003001		DV007003	AC164001	AC164002	AC164015			
PIC16C56	DV162003	DV003001		DV007003	AC164001	AC164002	AC164015			
PIC16C57	DV162003	DV003001		DV007003	AC164001	AC164002	AC164015			
PIC16C58A	DV162003	DV003001		DV007003	AC164001	AC164002	AC164015			
PIC16C554		DV003001		DV007003	AC164010	AC164010	AC164018			
PIC16C558		DV003001		DV007003	AC164010	AC164010	AC164018			
PIC16C61	DV162003	DV003001		DV007003	AC164010	AC164010				
PIC16C62A		DV003001		DV007003	AC164012	AC164017	AC164021			
PIC16C63	DV162002	DV003001		DV007003	AC164012	AC164017	AC164021			
PIC16C64A		DV003001		DV007003	AC164012			AC164013	AC164014	AC164020
PIC16C65A		DV003001		DV007003	AC164012			AC164013	AC164014	AC164020
PIC16C620	DV162003	DV003001		DV007003	AC164010	AC164010	AC164018			
PIC16C621	DV162003	DV003001		DV007003	AC164010	AC164010	AC164018			
PIC16C622	DV162003	DV003001		DV007003	AC164010	AC164010	AC164018			
PIC16C641		DV003001		DV007003	AC164012	AC164017				
PIC16C642		DV003001		DV007003	AC164012	AC164017				
PIC16C661		DV003001		DV007003	AC164012			AC164013	AC164014	AC164020
PIC16C662		DV003001		DV007003	AC164012			AC164013	AC164014	AC164020
PIC16C71	DV162003	DV003001		DV007003	AC164010	AC164010				
PIC16C710		DV003001		DV007003	AC164010	AC164010	AC164018			
PIC16C711		DV003001		DV007003	AC164010	AC164010	AC164018			
PIC16C715		DV003001		DV007003	AC164010	AC164010	AC164018			
PIC16C72		DV003001		DV007003	AC164012	AC164017	AC164021			
PIC16C73A		DV003001		DV007003	AC164012	AC164017				
PIC16C74A		DV003001		DV007003	AC164012			AC164013	AC164014	AC164020
PIC16F83		DV003001		DV007003	AC164010	AC164010				
PIC16F84		DV003001		DV007003	AC164010	AC164010				
PIC16C923		DV003001		DV007003	Planned			AC164022		AC164023
PIC16C924		DV003001		DV007003	Planned			AC164022		AC164023
PIC17C42A		DV003001		DV007003	AC174001			AC174002		AC174005
PIC17C43		DV003001		DV007003	AC174001			AC174002		AC174005
PIC17C44		DV003001		DV007003	AC174001			AC174002		AC174005
PIC17C756		DV003001 **		DV007003 **	Planned			AC174007* *		AC174008* *

Shaded area indicates not applicable.
**Contact Factory for Availability

© 1997 Microchip Technology Inc.

Demo Boards, Evaluation Kits Cross Reference and Ordering Information

Silicon Device	Model Name					
	PICDEM -1	PICDEM -2	PICDEM -3	PICDEM-14A	KEELOQ Evaluation Kit	Serial EEPROM Designer's Kit
Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #
24CXX/24LCXX						DV243001
25CXX/25LCXX						DV243001
93CXX/93LCXX						DV243001
HCS200					DM303001	
HCS300					DM303001	
HCS301					DM303001	
HCS360					DM303001	
HCS361					DM303001	
HCS509					DM303001	
HCS512					DM303001	
PIC12C508						
PIC12C509						
PIC12C671						
PIC12C672						
PIC14C000				DM143001		
PIC16C52	DM163001					
PIC16C54	DM163001					
PIC16C54A	DM163001					
PIC16C55	DM163001					
PIC16C56	DM163001					
PIC16C57	DM163001					
PIC16C58A	DM163001					
PIC16C554						
PIC16C558						
PIC16C61	DM163001					
PIC16C62A		DM163002				
PIC16C63		DM163002				
PIC16C64A		DM163002				
PIC16C65A		DM163002				
PIC16C620	DM163001					
PIC16C621	DM163001					
PIC16C622	DM163001					
PIC16C641						
PIC16C642						
PIC16C661						
PIC16C662						
PIC16C71	DM163001					
PIC16C710						
PIC16C711						
PIC16C715						

Shaded area indicates not applicable

© 1997 Microchip Technology Inc.

Demo Boards, Evaluation Kits Cross Reference and Ordering Information

Silicon Device	Model Name					
	PICDEM -1	PICDEM -2	PICDEM -3	PICDEM-14A	KEELOQ Evaluation Kit	Serial EEPROM Designer's Kit
Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #	Ordering #
PIC16C72		DM163002				
PIC16C73A		DM163002				
PIC16C74A		DM163002				
PIC16F83						
PIC16F84						
PIC16C923			DM163003			
PIC16C924			DM163003			
PIC17C42A	DM163001					
PIC17C43	DM163001					
PIC17C44	DM163001					
PIC17C756						

Shaded area indicates not applicable

© 1997 Microchip Technology Inc.

ON-LINE SUPPORT

Microchip Internet and BBS Connections

ON-LINE SUPPORT

Microchip provides two methods of on-line support. These are the Microchip BBS and the Microchip World Wide Web (WWW) site.

Use Microchip's Bulletin Board Service (BBS) to get current information and help about Microchip products. Microchip provides the BBS communication channel for you to use in extending your technical staff with microcontroller and memory experts.

To provide you with the most responsive service possible, the Microchip systems team monitors the BBS, posts the latest component data and software tool updates, provides technical help and embedded systems insights, and discusses how Microchip products provide project solutions.

The web site, like the BBS, is used by Microchip as a means to make files and information easily available to customers. To view the site, the user must have access to the Internet and a web browser, such as Netscape or Microsoft Explorer. Files are also available for FTP download from our FTP site.

Connecting to the Microchip Internet Web Site

The Microchip web site is available by using your favorite Internet browser to attach to:

www.microchip.com

The file transfer site is available by using an FTP service to connect to:

ftp://ftp.futureone.com/pub/microchip

The web site and file transfer site provide a variety of services. Users may download files for the latest Development Tools, Data Sheets, Application Notes, User's Guides, Articles and Sample Programs. A variety of Microchip specific business information is also available, including listings of Microchip sales offices, distributors and factory representatives. Other data available for consideration is:

- Latest Microchip Press Releases
- Technical Support Section with Frequently Asked Questions
- Design Tips
- Device Errata
- Job Postings
- Microchip Consultant Program Member Listing
- Links to other useful web sites related to Microchip Products

Connecting to the Microchip BBS

Connect worldwide to the Microchip BBS using either the Internet or the CompuServe® communications network.

Internet:

You can telnet or ftp to the Microchip BBS at the address:

mchipbbs.microchip.com

CompuServe Communications Network:

When using the BBS via the Compuserve Network, in most cases, a local call is your only expense. The Microchip BBS connection does not use CompuServe membership services, therefore you do not need CompuServe membership to join Microchip's BBS. There is no charge for connecting to the Microchip BBS.

The procedure to connect will vary slightly from country to country. Please check with your local CompuServe agent for details if you have a problem. CompuServe service allow multiple users various baud rates depending on the local point of access.

The following connect procedure applies in most locations.

1. Set your modem to 8-bit, No parity, and One stop (8N1). This is not the normal CompuServe setting which is 7E1.
2. Dial your local CompuServe access number.
3. Depress the **<Enter>** key and a garbage string will appear because CompuServe is expecting a 7E1 setting.
4. Type **+**, depress the **<Enter>** key and "`Host Name:`" will appear.
5. Type MCHIPBBS, depress the <Enter> key and you will be connected to the Microchip BBS.

In the United States, to find the CompuServe phone number closest to you, set your modem to 7E1 and dial (800) 848-4480 for 300-2400 baud or (800) 331-7166 for 9600-14400 baud connection. After the system responds with "`Host Name:`", type `NETWORK`, depress the **<Enter>** key and follow CompuServe's directions.

For voice information (or calling from overseas), you may call (614) 723-1550 for your local CompuServe number.

© 1997 Microchip Technology Inc.

On-Line Support

Microchip regularly uses the Microchip BBS to distribute technical information, application notes, source code, errata sheets, bug reports, and interim patches for Microchip systems software products. For each SIG, a moderator monitors, scans, and approves or disapproves files submitted to the SIG. No executable files are accepted from the user community in general to limit the spread of computer viruses.

Systems Information and Upgrade Hot Line

The Systems Information and Upgrade Line provides system users a listing of the latest versions of all of Microchip's development systems software products. Plus, this line provides information on how customers can receive any currently available upgrade kits.The Hot Line Numbers are:

1-800-755-2345 for U.S. and most of Canada, and

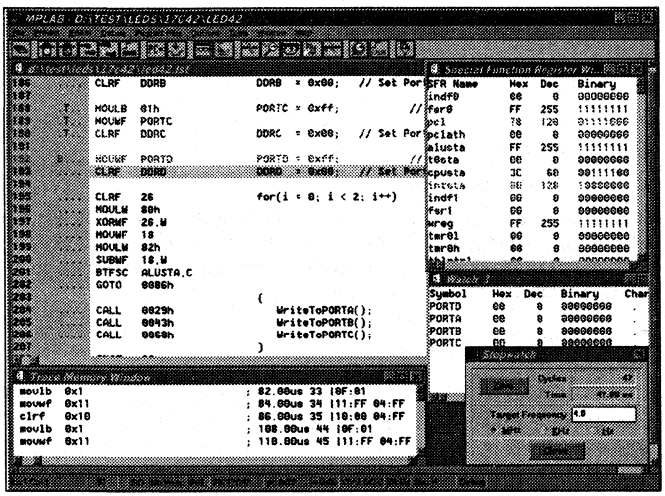
1-602-786-7302 for the rest of the world.

© 1997 Microchip Technology Inc.

MICROCHIP

MPLAB™

Integrated Development Environment Product Brief



MPLAB gives PICmicro MCU users the flexibility to edit, compile and debug from a single user interface.

MPLAB is a Windows®-based development platform for the Microchip Technology PICmicro™ one-time-programmable (OTP) MCU families. MPLAB Integrated Development Environment (IDE) offers a project manager and program text editor, a user-configurable toolbar containing four pre-defined sets and a status bar which communicates editing and debugging information.

MPLAB is the common user interface for Microchip development systems tools including MPLAB Editor, MPASM Assembler, MPLAB-SIM Software Simulator, PICMASTER® In-Circuit Emulator and PICSTART® Plus Development Programmer. Additional products may become available as add-on tools in the future.

The MPLAB desktop provides the development environment and tools for developing and debugging your application as a project, allowing you to quickly move between different development and debugging modes. With the MPLAB environment, you can debug your source code, automatically locate errors in source files for editing, debug with breakpoints based on internal register values, watch the program flow with MPLAB-SIM (software simulator) or PICMASTER, make timing measurements with a "stop watch," view variables in watch windows and find quick answers to questions from the MPLAB on-line help.

Features:

MPLAB Project Manager

- Organizes the different files that comprise your application firmware under one "Project"
- Allows you to create a project; add, edit or debug a source code file in a project; and build object files

MPLAB Editor

- Full-featured programmer's editor lets you write and edit firmware source files or other text files for PICmicro MCUs
- Creates re-usable source file templates for quick creation of new source files

MPASM Universal Assembler

- Has full-featured macro capabilities, conditional assembly and several source and listing formats
- Generates various object code formats to support Microchip development tools and third-party programmers without exiting MPLAB

MPLAB-SIM Software Simulator

- Discrete-event simulator integrated into the MPLAB IDE features debug capabilities including unlimited breakpoints, trace, examine/modify registers, watch variables and time-stamp
- Simulates the core functions as well as most peripherals of the PICmicro MCUs

11

© 1996 Microchip Technology Inc.

MPLAB IDE

Ordering Information:

Model Name:

MPLAB

Ordering Part Number:

SW007002

Tools Supported:

PICMASTER In-Circuit Emulator

PICSTART Plus Low-cost Programmer

PRO MATE®II Device Programmer

Host System Requirements:

PC with 386 or higher processor. Pentium™ recommended

4 MB Memory, 16 MB recommended

8 MB hard disk space, 20 MB recommended

VGA or Super VGA Monitor

Microsoft Windows 3.1 or greater

System Description:

MPLAB allows you to write, debug, and optimize the PICmicro MCU applications for firmware product designs.

MPLAB IDE software package includes the following:

- MPLAB Project Manager
- MPLAB Editor
- MPASM Universal Macro Assembler for the PICmicro MCUs and other language products supporting the Common Object Description file format
- MPLAB-SIM Software Simulator

MPLAB currently supports the following systems:

- PICMASTER In-Circuit Emulator
- PICSTART Plus Development Programmer

To order or obtain more information about MPLAB IDE, contact the Microchip sales office nearest you. MPLAB software may also be downloaded from Microchip's Bulletin Board System. User's can connect to the BBS through the CompuServe® communications network.

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers and Corporate Application Engineers as well as a multifaceted Bulletin Board System. Microchip's Internet home page can be reached at: http://www.microchip.com

"The Microchip Advantage"	
MPLAB™	Integrated Development Environment (IDE)
MPASM	Universal PICmicro macro-assembler
MPLAB-SIM	Software Simulator
MPLAB-C	C compiler for PICmicro microcontrollers
PICMASTER®	Full-featured modular in-circuit emulator
ICEPIC	Low-cost modular in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART® Plus	Entry-level development kit with programmer
PICSTART® Lite	Low-cost entry level development kit
fuzzyTECH®-MP	Fuzzy Logic development software
MP-DriveWay™	Application Code Generator
KEELOQ®	Code Hopping Devices

© 1996 Microchip Technology Inc.

MPASM Universal Assembler

Universal PICmicro™ MCU Assembler Software

This product brief describes the technical aspects of the PICmicro MCU Assembler. The MPASM Cross Assembler is a PC hosted symbolic assembler. It supports all microcontroller series, including the PIC16C5X, PIC16CXX and PIC17CXX families.

MPASM offers fully featured Macro capabilities, conditional assembly, and several source and listing formats. It generates various object code formats to support Microchip's development tools as well as third party programmers.

MPASM allows full symbolic debugging from the Microchip Universal Emulator System (PICMASTER®).

MPASM REQUIREMENTS

MPASM will run on any IBM PC/AT® or compatible computer running DOS 5.0 or later.

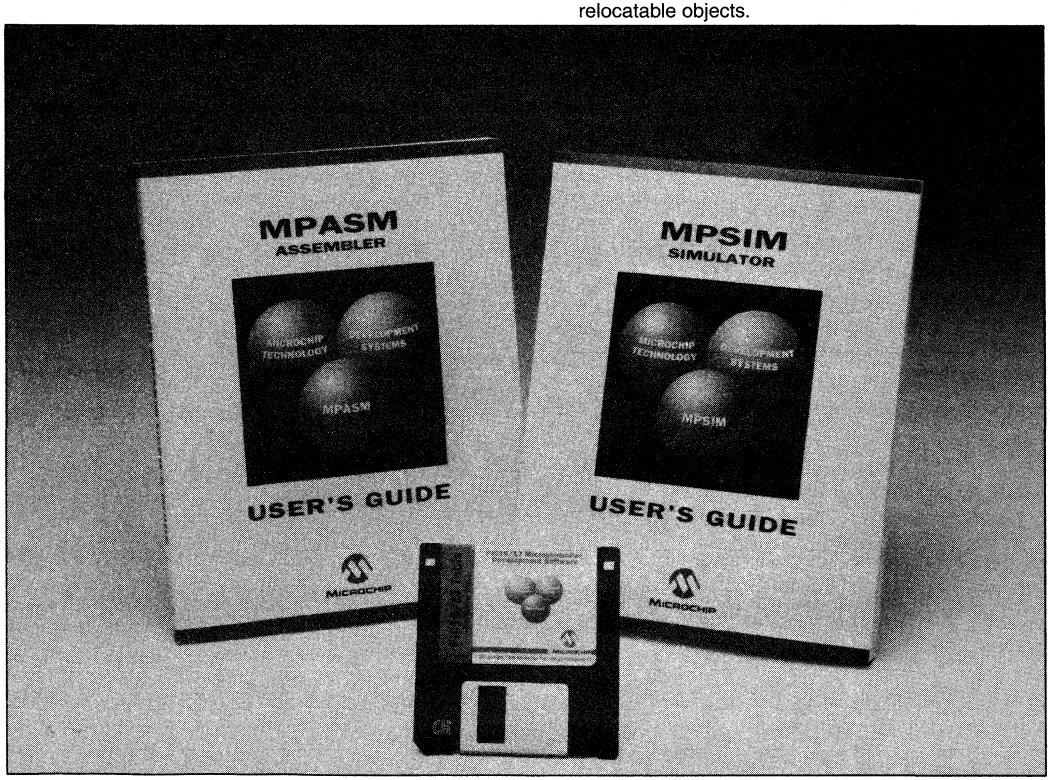
MPASM ASSEMBLER FEATURES

MPASM supports the 12-bit PIC16C5X, the 14-bit PIC16CXX, and the 16-bit PIC17CXX cores.

All instructions are single-word and single-cycle, except for branches, which execute in two cycles. Most instructions operate on one or more operands.

MPASM have the following features to assist in developing software for specific user applications:

- Provides translation of Assembler source code to object code for all Microchip microcontrollers.
- Macro Assembly Capability
- Provides Object, Listing, Symbol and special files required for debugging with one of the Microchip Emulator systems.
- Supports Hex (default), Decimal and Octal source and listing formats.
- Output formats: INHX8S, INHX8M, INHX32 and relocatable objects.



11

© 1995 Microchip Technology Inc.

MPASM DIRECTIVE LANGUAGE

MPASM provides a full featured directive language represented by four basic classes of directives:

- Data Directives are those that control the allocation of memory and provide a way to refer to data items symbolically, by meaningful names.
- Listing Directives control the MPASM listing display. They allow the specification of titles and subtitles, page ejects and other listing control.
- Control Directives permit sections of conditionally assembled code.
- Macro Directives control the execution and data allocation within macro body definitions.

MPASM INSTRUCTION SET

MPASM supports the entire instruction set of the PIC16C5X, PIC16CXX and PIC17CXX microcontrollers, as represented in the following four classes of instructions:

- Data Move Operations
- Arithmetic and Logical Operations
- Bit Manipulation Operations
- Special Control Operations

The Microchip microcontroller set is used to operate on data located in any of the file registers, including the I/O registers. There are:

- Data Transfer Operations
- Logical Operations
- Rotate Operations

MPASM provides bit level file register operations to manipulate and test individual bits in any addressable register, literal and control operations permitting operations on literals and branches to subroutines in program memory.

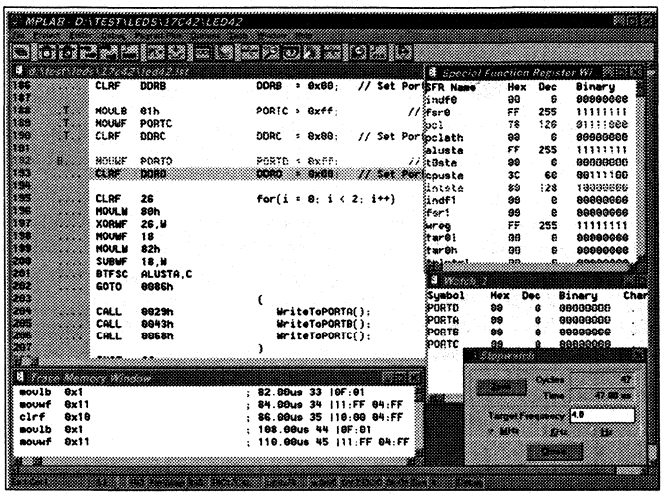
The Microchip microcontroller instruction sets allow read and write of special function registers such as the PC and status registers.

© 1995 Microchip Technology Inc.

MICROCHIP

MPLAB™SIM

Software Simulator Product Brief



MPLAB-SIM takes advantage of the power of MPLAB to provide an extremely powerful and cost effective debugging solution.

The MPLAB-SIM software simulator allows the user to isolate code problems and debug firmware designs on PICmicro™ MCU devices. It simulates the core functions as well as most of the peripherals of the PICmicro MCU families. It is particularly suitable for optimizing algorithms where real-time emulation is not required.

MPLAB is a Windows®-based development platform for the Microchip Technology PICmicro one-time-programmable (OTP) MCU families. MPLAB Integrated Development Environment (IDE) offers a project manager and program text editor, a user-configurable toolbar containing four pre-defined sets and a status bar which communicates editing and debugging information.

The MPLAB desktop provides the development environment and tools for developing and debugging your application as a project, allowing you to quickly move between different development and debugging modes. With the MPLAB environment, you can debug your source code, automatically locate errors in source files for editing, debug with breakpoints based on internal register values, watch the program flow with MPLAB-SIM (software simulator) or PICMASTER, make timing measurements with a "stop watch," view variables in watch windows and find quick answers to questions from the MPLAB on-line help.

Features:

MPLAB-SIM Software Simulator

- Discrete-event simulator integrated into the MPLAB IDE features debug capabilities including unlimited breakpoints, trace, examine/modify registers, watch variables and time-stamp
- Simulates the core functions as well as most peripherals of the PICmicro MCUs

MPLAB Project Manager

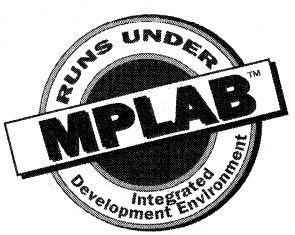
- Organizes the different files that comprise your application firmware under one "Project"
- Allows you to create a project; add, edit or debug a source code file in a project; and build object files

MPLAB Editor

- Full-featured programmer's editor lets you write and edit firmware source files or other text files for PICmicro MCUs
- Creates re-usable source file templates for quick creation of new source files

MPASM Universal Assembler

- Has full-featured macro capabilities, conditional assembly and several source and listing formats
- Generates various object code formats to support Microchip development tools and third-party programmers without exiting



11

© 1996 Microchip Technology Inc.

MPLAB-SIM

Ordering Information:

Model Name:

MPLAB

Ordering Part Number:

SW007002

Tools Supported:

MPLAB-SIM Software Simulator

PICMASTER In-Circuit Emulator

PICSTART Plus Low-cost Programmer

PRO MATE II Device Programmer

Host System Requirements:

PC with 386 or higher processor. Pentium™ recommended

4 MB Memory, 16 MB recommended

8 MB hard disk space, 20 MB recommended

VGA or Super VGA Monitor

Microsoft Windows 3.1 or greater

System Description:

MPLAB-SIM is a discrete event simulator software application designed to imitate operation of the PICmicro MCUs. It allows the user to debug software that will use any of these microcontrollers.

At any instruction boundary, you may examine and/or modify any data area within the processor, or provide external stimulus to any of the pins. MPLAB-SIM gives you a solid, low cost, source-level debug tool to help you through the early design verification stages of your project.

MPLAB-SIM runs under MPLAB. This allows you to write, debug, and optimize the PICmicro MCU applications for firmware product designs.

MPLAB IDE software package includes the following:

- MPLAB Project Manager
- MPLAB Editor
- MPASM Universal Macro Assembler for the PICmicro MCUs and other language products supporting the Common Object Description file format

To order or obtain more information about MPLAB-SIM, contact the Microchip sales office nearest you. MPLAB-SIM software may also be downloaded from Microchip's Bulletin Board System. User's can connect to the BBS through the CompuServe® communications network.

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers and Corporate Application Engineers as well as a multifaceted Bulletin Board System. Microchip's Internet home page can be reached at: http://www.microchip.com

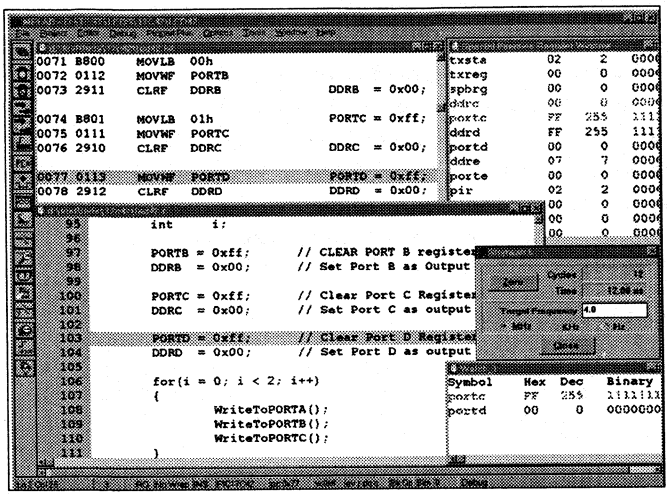
"The Microchip Advantage"	
MPLAB™	Integrated Development Environment (IDE)
MPASM	Universal PICmicro MCU macro-assembler
MPLAB-SIM	Software Simulator
MPLAB-C	C compiler for PICmicro MCU
PICMASTER®	Full-featured modular in-circuit emulator
ICEPIC	Low-cost modular in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART® Plus	Entry-level development kit with programmer
PICSTART® Lite	Low-cost entry level development kit
fuzzyTECH®-MP	Fuzzy Logic development software
MP-DriveWay™	Application Code Generator
KEELOQ®	Code Hopping Devices

© 1996 Microchip Technology Inc.

MICROCHIP

MPLAB™C

Universal C Compiler for PICmicro MCUs Product Brief



Features:

- MPLAB-compatible for source level debugging in an easy-to-use project environment
- Provides Object, Listing, Symbol and special files for debugging with MPLAB
- Supports interrupt routines
- Compares source code to target hardware definitions
- Header file register definitions compatible with MPASM Assembler
- Generates tight, efficient code
- Output formats: INHX8S, INHX8M and INHX32
- Has ability to embed assembly

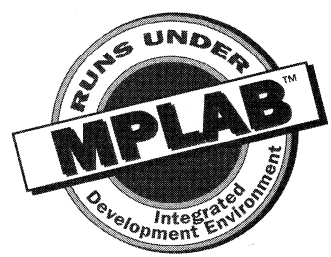
MPLAB-C provides powerful integration capabilities and ease of use not found with other compilers.

The **MPLAB-C** Universal C Compiler is a complete high-level language compiler for the Microchip Technology PICmicro™ families of 8-bit one-time-programmable (OTP) MCUs. Since MPLAB-C is compatible with Microchip's MPLAB Integrated Development Environment (IDE), it functions with the PICMASTER® in-circuit emulator and the MPLAB-SIM simulator. When using MPLAB-C with MPLAB, you get full source level debugging in an easy-to-use project environment which helps you reduce development time.

The MPLAB-C C compiler generates executable code directly from the compile process. There is no need for an extra step to assemble code generated by the compiler.

MPLAB-C allows you to write code for PICmicro MCU applications in a high level language. The detailed operation of the target processor is mostly hidden which means less time is devoted to considering the details of the processors's architecture. This feature also permits easy code portability between different PICmicro MCU devices – often as simple as changing one line of source code.

MPLAB-C will run on any IBM PC/AT® or compatible computer running DOS 5.0 or later, and it supports the 12-bit, 14-bit and 16-bit Microchip microcontroller cores. MPLAB-C is a rule based compiler with expert systems tailored to each of these platforms for optimal efficiency.



11

© 1996 Microchip Technology Inc.

Ordering Information:

Model Name:

MPLAB-C

Ordering Part Number:

SW006005

Devices Supported:

All PICmicro MCUs

Model Name:

MPLAB-C with MP-DriveWay™

Ordering Part Number:

SW006008

Devices Supported:

MPLAB-C supports all PICmicro MCUs.

MP-DriveWay supports: PIC16C54, PIC16C54A, PIC16C55, PIC16C56, PIC16C57, PIC16C58A, PIC16CR54, PIC16CR57A, PIC16CR58A, PIC16C64, PIC16C71, PIC16C74, PIC16C84, PIC17C42, PIC17C42A, PIC17C43, PIC17C44

System Requirements:

IBM PC/AT® or compatible computer

DOS 5.0 or later operating system

System Description:

The MPLAB-C Universal C Compiler comes complete with the MPLAB IDE. The IDE allows you to quickly move between different development and debugging modes, for example, you can quickly advance from software debugging with MPLAB-SIM to hardware debugging with PICMASTER.

MPLAB-C has implemented extensions to the C language to provide specific support for microcontroller environments, including:

- I/O Ports (read only, write only and read/write)
- Read only access to the processor status register
- Read/write access to the processor index register
- Read/write access to the processor working (W) register
- Interrupt vectors
- Fix variables and functions in memory

To order or obtain more information about MPLAB-C or any other Microchip product, contact the Microchip sales office nearest you.

Customer Support:

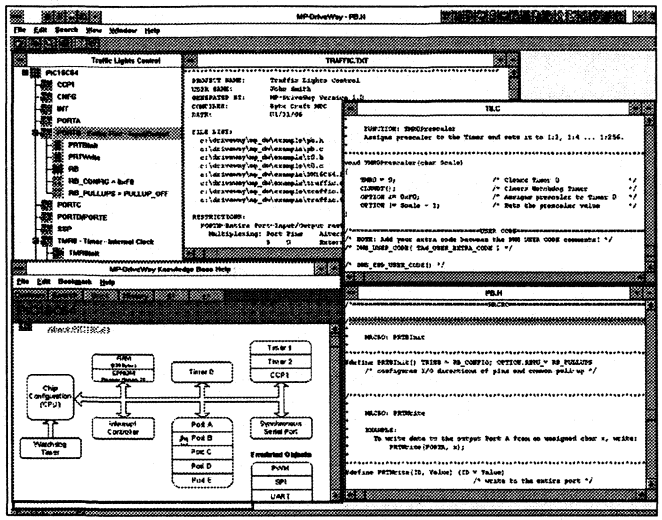
Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers and Corporate Application Engineers as well as a multifaceted Bulletin Board System. Microchip's Internet home page can be reached at: http://www.microchip.com

"The Microchip Advantage"	
MPLAB™	Integrated Development Environment (IDE)
MPASM	Universal PICmicro macro-assembler
MPLAB-SIM	Software Simulator
MPLAB-C	C compiler for PICmicro MCU
PICMASTER®	Full-featured modular in-circuit emulator
ICEPIC	Low-cost modular in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART® Plus	Entry-level development kit with programmer
PICSTART® Lite	Low-cost entry level development kit
fuzzyTECH®-MP	Fuzzy Logic development software
MP-DriveWay™	Application Code Generator
KEELOQ®	Code Hopping Devices

© 1996 Microchip Technology Inc.

MP-DRIVEWAY™

Application Code Generator Product Brief



MP-DriveWay generates 'C' code for PICmicro™ MCUs.

MP-DriveWay is an easy to use Windows®-based Application Code Generator for Microchip Technology's 8-bit, one-time-programmable (OTP) PICmicro MCUs. With MP-DriveWay you can visually configure all the peripherals in a PICmicro MCU device and, with a click of the mouse, generate all the initialization and many functional code modules in C language.

The output of MP-DriveWay is fully compatible with Microchip's MPLAB-C compiler. The code produced is highly modular and allows easy integration of your own code. In fact, MP-DriveWay is intelligent enough to maintain your code through subsequent code generation.

MP-DriveWay derives code output from an extensive knowledge base. For each peripheral, the knowledge base contains an array of driver routines. For example, the USART module can be configured to be full duplex or half duplex, 8- or 9-bit, interrupt driven or polled and so on. MP-DriveWay, further allows you to specify desired operating parameters, such as baud rate. This information is used to generate all the appropriate control register values which are then plugged into the initialization code.

MP-DriveWay also generates "emulated" or "virtual" peripherals. Peripheral functions such as USART or PWM that may not exist on a selected PICmicro MCU are implemented entirely in software.

Features:

- Application Code Generator for PICmicro MCUs
- Windows-based intuitive user interface
- Describe peripheral operation at a high level
- Generates initialization and functional code
- Generates C code, compatible with Microchip's MPLAB-C
- Built in knowledge base for all peripherals
- Code generated is fully tested and documented reducing your debug time
- Specify operating parameters (such as baud rate)
- Emulated Peripherals
- IDS: Interactive Data Sheet, is an innovative on-line help
- Reduce your learning curve on a new device
- Use DriveWay to prototype quickly

11

© 1996 Microchip Technology Inc.

MP-DriveWay

Ordering Information:

Model Name:

MP-DriveWay

Ordering Part Number:

SW006006

Devices Supported:

PIC16C54, PIC16C54A,

PIC16C55, PIC16C56,

PIC16C57, PIC16C58A,

PIC16CR54, PIC16CR57A, PIC16CR58A

PIC16C64

PIC16C71, PIC16C74

PIC16C84

PIC17C42A, PIC17C43, PIC17C44

Model Name:

MP-DriveWay with MPLAB-C

Ordering Part Number:

SW006008

Devices Supported:

See device listing above for

MP-DriveWay support.

MPLAB-C supports all PICmicro MCUs.

Host System Requirements:

386 based PC or greater

Microsoft Windows 3.1 or higher

4 MB RAM, 5 MB free disk space

3.5 inch diskette drive

MP-DriveWay has an extensive and unique on-line interactive help called Integrated Data Sheet (IDS). It is geared towards engineers developing code. With IDS you can find all the information you need for any peripheral all in one place.

MP-DriveWay generates fully tested and documented code, reducing your debug time. Use of pre-developed code segments from the knowledge base results in highly efficient code output.

MP-DriveWay enhances your productivity. As a new user, you can reduce your learning curve. As an expert user, it allows you to prototype applications faster.

MP-DriveWay is a result of partnership effort between by AISYS Intelligent Systems Ltd. of Israel and Microchip.

To order or obtain more information about MP-DriveWay or any other Microchip product, contact the Microchip sales office nearest you.

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers and Corporate Application Engineers as well as a multifaceted Bulletin Board System. Microchip's Internet home page can be reached at: http://www.microchip.com

"The Microchip Advantage"	
MPLAB™	Integrated Development Environment (IDE)
MPASM	Universal PICmicro MCU macro-assembler
MPLAB-SIM	Software Siumulator
MPLAB-C	C compiler for PICmicro MCUs
PICMASTER®	Full-featured modular in-circuit emulator
ICEPIC	Low-cost modular in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART® Plus	Entry-level development kit with programmer
PICSTART® Lite	Low-cost entry level development kit
fuzzyTECH®-MP	Fuzzy Logic development software
MP-DriveWay™	Application Code Generator
KEELOQ®	Code Hopping Devices

© 1996 Microchip Technology Inc.

MICROCHIP

*fuzzy*TECH®MP

Fuzzy Logic Development System for PICmicro MCUs

*fuzzy*TECH-MP FOR MICROCHIP PICmicro MCUs

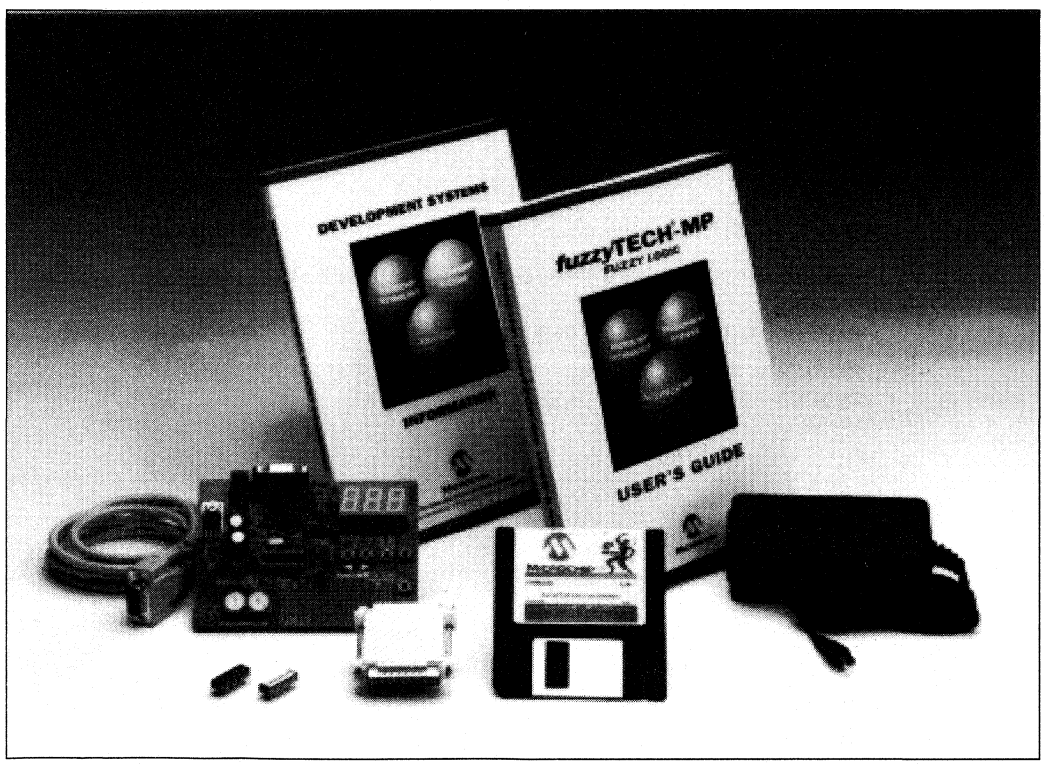
This product brief describes the technical aspects of the *fuzzy*TECH-MP Fuzzy Logic Development System for PICmicro MCUs developed by INFORM Software Corporation specifically for Microchip.

The *fuzzy*TECH-MP Development System comes in two versions. The first, the Explorer, contains everything you need to gain a comprehensive working knowledge about fuzzy-logic system design. It is easy-to-use, all graphic editors and tools guide you step-by-step through the development phases of fuzzy systems. The Explorer supports two input variables and one output variable.

The full-featured *fuzzy*TECH-MP Edition offers all of the capabilities of the Explorer, plus it has the additional flexibility of eight input variables and four output variables for designing more complex systems. The full features are enabled with a hardware key lock attached to the parallel port of the PC.

Included in both versions is *fuzzy*LAB™, a fully functional demonstration board, to give customers hand-on experience with fuzzy logic systems implementation. *fuzzy*LAB is a simple heating thermostat consisting of a PWM-controlled resistor configured to heat a thermistor to a preset temperature. Using the two fuzzy algorithms provided, a designer can set a target temperature and observe the thermostat response to the set point.

Both systems generate assembly code compatible with the MPASM, Microchip's Universal Assembler, that can be integrated into your application. Examining this code provides you with further insights into the fabrics of fuzzy logic systems.



11

© 1995 Microchip Technology Inc.

*fuzzy*TECH-MP System Requirements

*fuzzy*TECH-MP will run on any IBM PC® (386 or higher) or compatible computer, running DOS 4.1 or later, and Microsof®t Windows® 3.0 or later. Because *fuzzy*TECH-MP makes extensive use of graphics, a color graphic monitor (VGA) is required, and higher resolutions of 800 x 600 or 1024 x 768 are recommended.

What is Fuzzy Logic?

Fuzzy logic is a technology that enhances mode-based system designs using both intuition and engineering heuristics. Fuzzy logic uses elements of everyday language to represent desired system behavior, thus circumventing the need for rigorous mathematical modeling.

It is an efficient way of designing, optimizing and maintaining highly complex systems transparently.

Fuzzy Logic Applications

Fuzzy logic finds its home in unique applications:

- When no adequate mathematical model for a given problem is readily apparent.
- When non-linearities, time constraints or multiple parameters exist.
- When engineering know-how about the given problem is available or can be acquired during the design process.

The *fuzzy*TECH-MP Implementation

*fuzzy*TECH-MP provides the following standard features:

- Windows Compatible with full graphical user interface
- 8-Input variables (2 for the Explorer version)
- 4-Output variables (1 for the Explorer version)
- 8-Bit resolution on input and output variables
- 16-Bit computation resolution for the PIC16CXX and PIC17CXX microcontrollers
- No theoretical limit on rules, antecedents and linguistic conjunctions (chip limitations will place a practical limit on these)
- MAX-MIN and MAX-DOT inference methods
- CoM and MoM defuzzification methods
- MPASM Compatible
- PICMASTER™ Compatible

*fuzzy*TECH-MP

*fuzzy*TECH-MP is available directly from Microchip Technology and its authorized distributors. Contact your local sales office for more information.

SALES AND SUPPORT

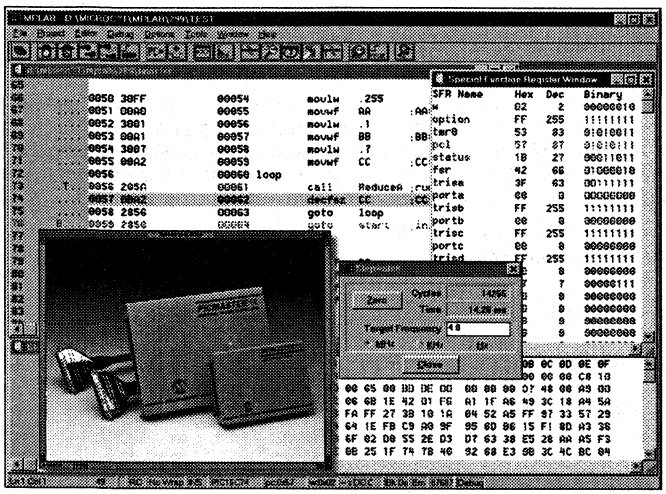
To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
DV005001	*fuzzy*TECH-MP EXPLORER
DV005002	*fuzzy*TECH-MP EDITION

© 1995 Microchip Technology Inc.

PICMASTER®CE

CE Compliant In-Circuit Emulator Product Brief



CE compliant PICMASTER offers design engineers maximum power, flexibility and rapid design turn-around!

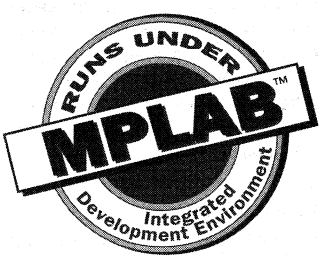
The PICMASTER-CE in-circuit emulator systems from Microchip Technology provide a high-quality, universal platform for emulating the entire line of Microchip 8-bit, one-time-programmable (OTP) microcontrollers. PICMASTER-CE is identical in user functionality with the popular PICMASTER emulators, but features additional enhancements.

PICMASTER-CE is a high-performance, real-time in-circuit emulator. Included in the complete system are the emulator, MPLAB™ integrated development environment, MPASM full-featured macro assembler, and MPLAB-SIM, an unusually sophisticated simulator program. Systems may be purchased that also include a device programmer (see ordering information on reverse side). Sample programs are provided to help quickly familiarize the user with the development system and the PICmicro 8-bit OTPMCU families. The MPLAB IDE user interface features a built-in text editor, project management, and full integration with MPASM and high level languages that support the Common Object Description format (i.e. MPLAB-C and MPC).

PICMASTER-CE meets or exceeds the stringent Electromagnetic Compatibility requirements known as CE compliance directives established by the European Union (EU) countries. These include limiting radiated emission, improving susceptibility to radiated emission and improving susceptibility to Electrostatic Discharge (ESD).

Features:

- High-performance PC-based development system for PICmicro MCUs
- Real-time in-circuit emulation
- Program memory emulation and memory mapping capability up to 64k words
- Real-time trace
- Unlimited hardware breakpoints
- External break with AND/OR capability with identical breakpoints
- Trigger out
- Multi-processor emulation
- Highly shielded cable to connect pod and prober
- Supported by MPLAB IDE for Windows
- Assembly and C source level debugging
- Includes MPLAB Project Manager
- Includes MPLAB Editor
- Includes MPASM Universal Assembler
- Includes MPLAB-SIM Simulator



11

© 1996 Microchip Technology Inc.

Ordering Information:

Complete Systems with PRO MATE II programmer:

Ordering Part #:	Devices Supported:
EM147101	PIC14000
EM167101	PIC123C508, 509, PIC16C52, 54, 54A, 55, 56, 57, 58A
EM167103	PIC16C62, 63, 64, 65, 72, 73, 74
EM167105	PIC16C71, 710, 711
EM167107	PIC16F83, 84
EM167109	PIC16C620, 621, 622
EM167111	PIC16C923, 924
EM167113	PIC16C554, 558
EM167115	PIC16C16C642, 662
EM167117	PIC16C715
EM167121	PIC16C66, 67, 76, 77
EM177107	PIC17C42A, 43, 44

Complete Systems without PRO MATE II programmer:

Ordering Part #:	Devices Supported:
EM147102	PIC14000
EM167102	PIC16C52, 54, 54A, 55, 56, 57, 58A,
EM167104	PIC16C62, 63, 64, 65, 72, 73, 74
EM167106	PIC16C71, 710, 711
EM167108	PIC16F83, 84
EM167110	PIC16C620, 621, 622
EM167112	PIC16C923, 924
EM167114	PIC16C554, 558
EM167116	PIC16C16C642, 662
EM167118	PIC16C715
EM167121	PIC16C66, 67, 76, 77
EM177108	PIC17C42A, 43, 44

Host System Requirements:

- PC-compatible computer: 386DX, 486 or PENTIUM™ based with ISA or EISA bus
- EGA, VGA, 8514/A, Hercules graphic card (EGA or higher recommended)
- MS-DOS®/PC-DOS® v. 5.0 or greater
- Microsoft Windows 3.1 or greater in 386 enhanced mode
- 4 MB RAM, 4 MB free disk space
- 3.5 inch diskette drive
- One 8-bit PC/AT (ISA) I/O expansion slot (half size)

System Description:

The PICMASTER-CE emulator systems feature real-time instruction emulation with program memory emulation and memory mapping capability up to 64K words. Instruction execution can be mapped into either emulation memory or user prototype memory. An unlimited number of hardware breakpoints can be set anywhere in the program memory.

The multiprocessor emulation capability allows for up to four PICMASTER-CE emulators to be synchronized on a single PC. PICMASTER-CE is capable of full symbolic debugging with symbolic display and alter of all register files, special purpose registers, stack registers and bank registers.

Interchangeable target probes allow the system to be easily reconfigured for emulation of different processors. PIC16C5X probes operate to 20 MHz, PIC17CXX to 16 MHz and PIC16CXX to 10 MHz. The universal architecture of the PICMASTER-CE allows expansion to support all new microcontroller architectures with data and program memory paths to 16 bits.

To order or obtain more information about PICMASTER-CE or any other Microchip product, contact the Microchip sales office nearest.

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers and Corporate Application Engineers as well as a multifaceted Bulletin Board System. Microchip's Internet home page can be reached at: http://www.microchip.com

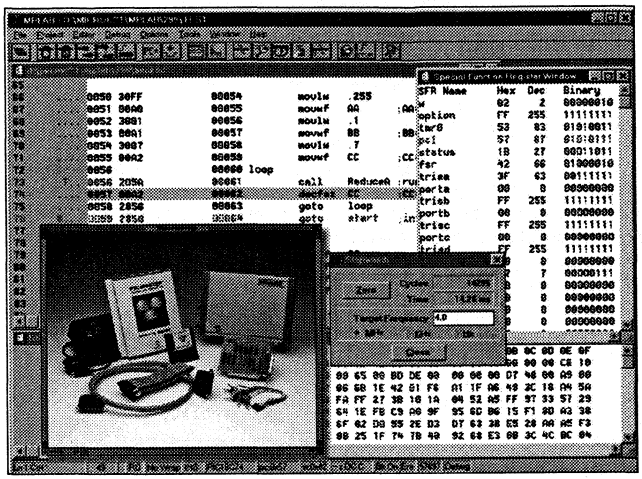
"The Microchip Advantage"	
MPLAB™	Integrated Development Environment (IDE)
MPASM	Universal PICmicro MCU macro-assembler
MPLAB-SIM	Software Simulator
MPLAB-C	C compiler for PICmicro MCUs
PICMASTER®	Full-featured modular in-circuit emulator
ICEPIC	Low-cost modular in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART® Plus	Entry-level development kit with programmer
PICSTART® Lite	Low-cost entry level development kit
fuzzyTECH®-MP	Fuzzy Logic development software
MP-DriveWay™	Application Code Generator
KEELOQ®	Code Hopping Devices

© 1996 Microchip Technology Inc.

MICROCHIP

PICMASTER®

Full-Featured Modular In-Circuit Emulator



FEATURES:

- Hi-performance PC-based development system for PICmicro MCUs

Emulator System Components:

- Host Interface Card connects emulator system to a PC compatible system
- The Emulator Control Pod contains emulation memory, trace memory, event and cycle timers and trace/ breakpoint logic
- A Target-specific Emulator Probe configures the universal system for emulation of a specific microcontroller
- MPLAB IDE provides the user with quick editing, debugging, optimization and control of the system under emulation

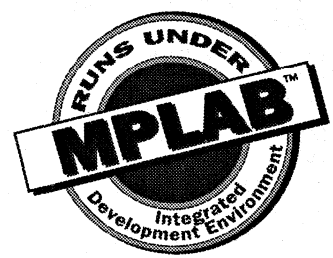
PICMASTER development systems offer maximum power, flexibility and rapid design turn-around!

The PICMASTER systems from Microchip Technology are full-featured, professional quality emulator systems. This flexible in-circuit emulator system provides a high-quality, universal platform for emulating the entire line of Microchip 8-bit, one-time-programmable (OTP) microcontrollers. PICMASTER systems are sold worldwide, with a CE compliant model available for European Union (EU) countries.

Provided with the PICMASTER emulator systems are the MPLAB™ integrated development environment; a high performance, real-time In-Circuit Emulator; an EPROM programmer unit (PRO MATE II); a macro assembler program (MPASM); and an unusually sophisticated simulator program (MPLAB-SIM). Sample programs are provided to help quickly familiarize the user with the development system and the PICmicro 8-BIT OTP MCU families. The MPLAB IDE user interface features a built-in text editor, project management, and full integration with MPASM and high level languages that support the Common Object Description format (i.e. MPLAB-C and MPC).

The PICMASTER emulator systems feature real-time instruction emulation with program memory emulation and memory mapping cabability up to 64K words. Instruction execution can be mapped into either emulation memory or user prototype memory. An unlimited number of hardware breakpoints can be set anywhere in the program memory.

The multiprocessor emulation capability allows for up to four PICMASTER emulators to be synchronized on a single PC. PICMASTER is capable of full symbolic debugging with symbolic display and alter of all register files, special purpose registers, stack registers and bank registers.



11

© 1996 Microchip Technology Inc.

Ordering Information:

Complete Systems with PRO MATE II programmer:

Ordering Part #:	Devices Supported:
EM147101	PIC14000
EM167015	PIC12C508. 509, PIC16C52, 54, 54A, 55, 56, 57, 58A
EM167021	PIC16C61
EM167023	PIC16C620, 621, 622
EM167031	PIC16C923, 924
EM167025	PIC16C62, 63, 64, 65, 72, 73, 74
EM167027	PIC16C71, 710, 711
EM167029	PIC16F83, 84
EM167109	PIC16C620, 621, 622
EM167033	PIC16C554, 558
EM167035	PIC16C642, 662
EM167037	PIC16C715
EM167041	PIC16C66, 67, 76, 77
EM177007	PIC17C42A, 43, 44

Complete Systems without PRO MATE II programmer:

Ordering Part #:	Devices Supported:
EM147102	PIC14000
EM167102	PIC12C508, 509, PIC16C52, 54, 54A, 55, 56, 57, 58A,
EM167104	PIC16C62, 63, 64, 65, 72, 73, 74
EM167106	PIC16C71, 710, 711
EM167108	PIC16F83, 84
EM167110	PIC16C620, 621, 622
EM167032	PIC16C923, 924
EM167034	PIC16C554, 558
EM167036	PIC16C16C642, 662
EM167038	PIC16C715
EM167042	PIC16C66, 67, 76, 77
EM177108	PIC17C42A, 43, 44

Host System Requirements:

- PC-compatible computer: 386DX, 486 or PENTIUM™ based with ISA or EISA bus
- EGA, VGA, 8514/A, Hercules graphic card (EGA or higher recommended)
- MS-DOS®/PC-DOS® v. 5.0 or greater
- Microsoft Windows 3.1 or greater in 386 enhanced mode
- 4 MB RAM, 4 MB free disk space
- 3.5 inch diskette drive
- One 8-bit PC/AT (ISA) I/O expansion slot (half size)

System Description:

The PICMASTER Universal In-Circuit Emulator Systems are intended to provide the product development engineer with a complete microcontroller design tool set for all microcontrollers in the PICmicro MCU family.

Interchangeable target probes allow the system to be easily reconfigured for emulation of different processors. The universal architecture of the PICMASTER allows expansion to support all new microcontroller architectures with data and program memory paths to 16 bits.

To order or obtain more information about PICMASTER or any other Microchip product, contact the Microchip sales office nearest.

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers and Corporate Application Engineers as well as a multifaceted Bulletin Board System. Microchip's Internet home page can be reached at: http://www.microchip.com

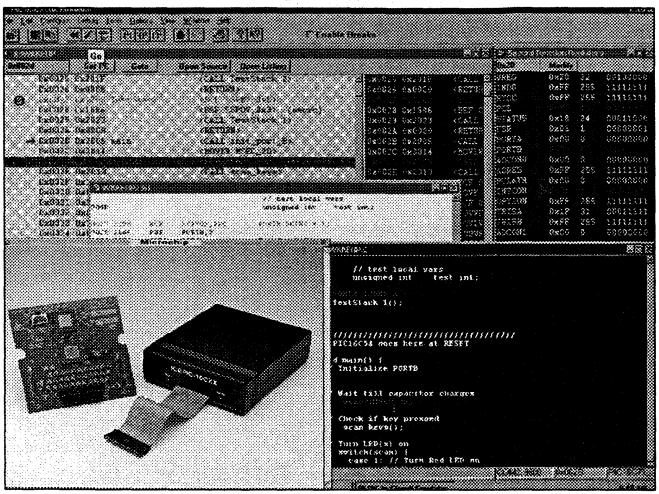
"The Microchip Advantage"	
MPLAB™	Integrated Development Environment (IDE)
MPASM	Universal PICmicro MCU macro-assembler
MPLAB-SIM	Software Simulator
MPLAB-C	C compiler for PICmicro MCUs
PICMASTER®	Full-featured modular in-circuit emulator
ICEPIC	Low-cost modular in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART® Plus	Entry-level development kit with programmer
PICSTART® Lite	Low-cost entry level development kit
fuzzyTECH®-MP	Fuzzy Logic development software
MP-DriveWay™	Application Code Generator
KEELOQ®	Code Hopping Devices

© 1996 Microchip Technology Inc.

MICROCHIP

ICEPIC™

Low-Cost PIC16CXXX In-Circuit Emulator Product Brief



ICEPIC: Affordable PIC16CXXX In-Circuit Emulation Solution.

ICEPIC is a low-cost in-circuit emulation solution for the Microchip Technology PIC16C5X and PIC16CXXX families of 8-bit one-time-programmable (OTP) microcontrollers. The modular system can support different subsets of PIC16C5X or PIC16CXXX products through the use of interchangeable personality modules or daughter boards. The emulator is capable of emulating without target application circuitry being present.

ICEPIC is designed to operate on PC-compatible machines ranging from 286-AT® systems through the new Pentium™ based machines. The ICEPIC development software runs under Microsoft Windows® 3.X environment, allowing the operator access to a wide range of supporting software accessories.

The ICEPIC development software provides a user-friendly operating environment with an easy-to-use toolbar; unlimited number of breakpoints; single, multiple and procedure step; ability to display and modify any register; user-selectable processor speeds via an oscillator module; full context-sensitive help and an RS-232 serial port.

ICEPIC is fully compatible with Microchip's MPASM Universal Assembler and MPLAB-C Compiler.

Features:

- Real time, non-intrusive emulation of PIC16C5X and PIC16CXXX microcontrollers
- 8K words of emulation memory
- Full speed, real time emulation to 20 MHz for PIC16C5X family
- Up to 10 MHz real time emulation for PIC16CXXX family
- Microsoft Windows® compatible
- Source level debug capability in assembly or C
- Symbolic debug capability
- 8K hardware breakpoints
- Custom watch points
- PC communication via serial interface at speeds up to 57K baud
- Display and modify any register (Program or Data)
- User selectable processor speeds (via oscillator module)

11

© 1996 Microchip Technology Inc.

Ordering Information:

Model Name	Ordering Part Number	Devices Supported
Complete Emulator Systems:		
ICEPIC5X	EM167201	16C54/54A, 55, 56, 57, 58A
ICEPIC62X	EM167202	16C620, 621, 622
ICEPIC74A	EM167207	16C64, 62, 16C74, 73, 65, 63
ICEPIC71	EM167205	16C71, 61
ICEPIC84	EM167206	16C84, F83, F84
ICEPIC16C92X	EM167210	16C923, 924
Daughter Boards:		
ICEPICDB5X	AC165201	16C54/54A, 55, 56, 57, 58A
ICEPICDB62X	AC165202	16C620, 621, 622
ICEPICDB74A	AC165203	16C64, 62, 16C74, 73, 65, 63
ICEPICDB71	AC165205	16C71, 61
ICEPICDB84	AC165206	16C84, F83, F84
ICEPICDB92X	AC165210	16C923, 924
Base Unit Only Without Daughter Board:		
ICEPIC POD	EM167200	All

System Description:

The low-cost PC-based ICEPIC In-Circuit Emulator system comes with an emulator unit (mother board), power supply, RS-232 cable, probe header cable(s) to connect to the application circuit, and one device-specific personality daughter board.

These interchangeable personality modules or daughter boards are contained with the mother board within one housing, connecting to the target application via a connector cable that extends from the housing. The mother board incorporates the common emulation logic while the daughter board is for device-specific emulator logic. This economical system allows the user to purchase a new daughter board for a new processor group as needed, at approximately 30% of the full system cost.

ICEPIC was designed by NEOSOFT Inc. and is manufactured under license by RF Solutions Ltd. To order or obtain more information about ICEPIC or any other Microchip product, contact the Microchip sales office nearest you.

Customer Support:

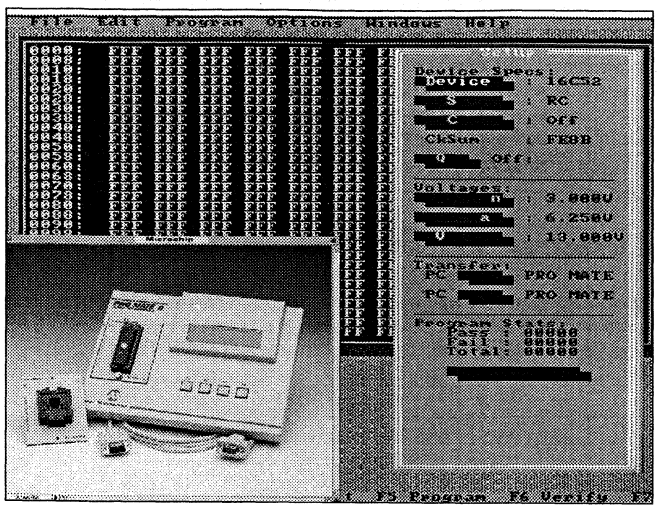
Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers and Corporate Application Engineers as well as a multifaceted Bulletin Board System. Microchip's Internet home page can be reached at: http://www.microchip.com

"The Microchip Advantage"	
MPLAB™	Integrated Development Environment (IDE)
MPASM	Universal PICmicro MCU macro-assembler
MPLAB-SIM	Software Simulator
MPLAB-C	C compiler for PICmicro MCUs
PICMASTER®	Full-featured modular in-circuit emulator
ICEPIC	Low-cost modular in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART® Plus	Entry-level development kit with programmer
PICSTART® Lite	Low-cost entry level development kit
fuzzyTECH®-MP	Fuzzy Logic development software
MP-DriveWay™	Application Code Generator
KEELOQ®	Code Hopping Devices

© 1996 Microchip Technology Inc.

PRO MATE®II

Universal Device Programmer Product Brief



CE Compliant PRO MATE II makes programming easy.

The PRO MATE II device programmer tool allows development engineers to program user software into Microchip Technology's entire line of PICmicro™ 8-bit one-time-programmable (OTP) MCUs. PRO MATE II is CE compliant, meaning it meets or exceeds all the directives for safety, emissions, ESD and susceptibility (to radiated emission) requirements set forth by the European Union (EU) countries.

The PRO MATE II device programmer is designed to be robust and reliable with: enhanced socket module alignment with four auto alignment pins; three levels of over-current protection and superior electrostatic discharge immunity for rugged environments; a small and compact univeral IEC power supply and improved LCD display and buttons.

The PRO MATE II device programmer is easy to use and operates either as a stand-alone unit or in conjunction with a PC-compatible host system. This time-saving tool comes complete with all the accessories needed to connect to a host system including interface cables and a universal input power supply.

In addition to the programmer unit, the PRO MATE II system contains Microchip's popular discrete event software simulator (MPLAB-SIM), a universal PICmicro macro assembler (MPASM), and host operating software which offers file display and editing and transfer to and from the programming unit.

Features:

- Programs EPROM and/or EEPROM program and data memory for all Microchip PICmicro MCUs
- Operates as stand-alone or with PC-host system
- Three operating modes:
  - Host Mode
  - Safe Mode
  - Stand-Alone Mode
- Complete line of interchangeable socket modules supports all package options (sold separately)
- Universal platform can quickly and easily support future Microchip products
- SQTP^{SM} serialization adds a unique serial number to each device programmed (in PC host mode)
- Safe Mode prevents accidental corruption of master code
- Saves and restores "environment" including all device settings, object file and more
- Indexed on-line help
- Complete documentation including all User's Guides

RUNS UNDER MPLAB™ Integrated Development Environment

11

© 1996 Microchip Technology Inc.

PRO MATE II

Ordering Information:

Model Name:

PRO MATE II

Ordering Part Number:

DV007003

Devices Supported:

ALL

PIC12CXXX

PIC14X000

PIC16C5X

PIC16C6X

PIC16C7X

PIC16C8X

PIC16C92X

PIC17CXXX

HCS200

HCS300

24CXX

93CXX

Socket Modules are sold separately. For a complete list, please contact Microchip or refer to the Development Systems Ordering Guide (DS30177).

System Description:

Microchip's PRO MATE II makes it easy to program the company's entire line of 8-bit OTP RISC microcontrollers, operating either as a stand-alone unit or in conjunction with a PC-compatible host system. When connected to a host system, PRO MATE II provides an exceptionally user-friendly interface to give the developer complete control over the programming session.

The new PRO MATE II software provides many user interface features. These include a "safe mode" targeting production environments where accidental corruption of master code is prevented; the ability to save and restore "environment," including all device settings, object file and more; and complete indexed on-line help. The PRO MATE II system also contains Microchip's popular discrete event software simulator (MPLAB-SIM) and a universal PICmicro macro assembler (MPASM).

To order or obtain more information about PRO MATE II or any other Microchip product, contact the Microchip sales office nearest you.

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers and Corporate Application Engineers as well as a multifaceted Bulletin Board System. Microchip's Internet home page can be reached at: http://www.microchip.com

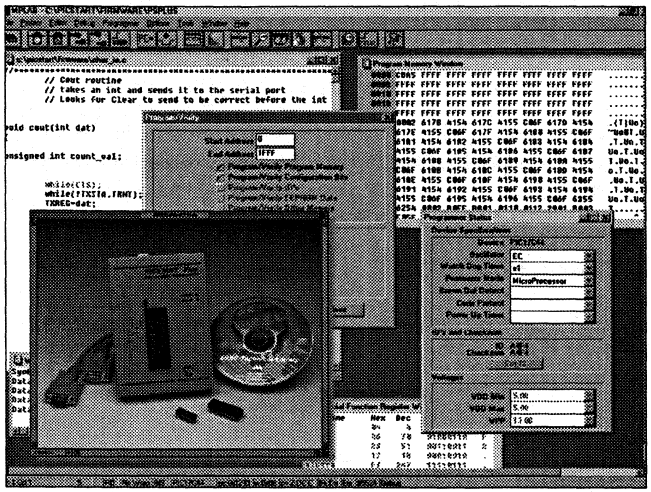
"The Microchip Advantage"	
MPLAB™	Integrated Development Environment (IDE)
MPASM	Universal PICmicro macro-assembler
MPLAB-SIM	Software Simulator
MPLAB-C	C compiler for PICmicro MCUs
PICMASTER®	Full-featured modular in-circuit emulator
ICEPIC	Low-cost modular in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART® Plus	Entry-level development kit with programmer
PICSTART® Lite	Low-cost entry level development kit
fuzzyTECH®-MP	Fuzzy Logic development software
MP-DriveWay™	Application Code Generator
KEELOQ®	Code Hopping Devices

© 1996 Microchip Technology Inc.

MICROCHIP

PICSTART®PLUS

Low Cost Development Kit Product Brief



PICSTART Plus makes designing with Microchip MCUs simple and affordable.

The **PICSTART Plus** development system from Microchip Technology provides the product development engineer with a highly-flexible low-cost microcontroller design tool set for all PIC14000, PIC16C5X, PIC16CXX and PIC17CXX 8-bit one-time-programmable (OTP) devices (DIP packages up to 40 pins).

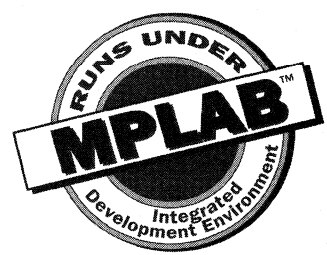
The PICSTART Plus development system operates on any PC-compatible machine running under the Windows® 3.1 operating system. PICSTART Plus is easy-to-use and features Microchip's highly acclaimed MPLAB™ Integrated Development Environment, with its built-in editor, assembler and Windows-based MPLAB-SIM simulator. The PICSTART Plus development system includes full documentation, software, development programmer and a device sample.

The CE compliant PICSTART Plus development programmer features a molded plastic enclosure and special circuit design techniques to enhance ESD protection. PICSTART Plus is a development programmer and is not recommended for use in a production environment.

Sample software programs are provided to help the developer quickly become familiar with the PICSTART Plus development system and with Microchip's microcontroller families. The PICSTART Plus system also includes Microchip's new CD-ROM containing a complete databook and Embedded Control Handbook (Application notes).

Features:

- Operates with PC-compatible host system running Windows under MPLAB™ environment
- Reads, programs, verifies EPROM and EEPROM program and data memory
- Reads, programs, verifies all configuration bits
- Programs and verifies an address range
- Displays, edits, and transfers device contents to and from programmer unit
- MPLAB Project support to automatically download object file to PICSTART Plus
- MPASM Assembler translates assembler source code to object code for all PICmicro devices
- MPLAB-SIM Windows-based simulator designed to model operation of all PICmicro MCUs
- Complete with RS-232 cable, 9 volt universal IEC power supply and power cable
- PICmicro MCU device sample
- Complete documentation, User's Guides and CD-ROM



11

© 1996 Microchip Technology Inc.

PICSTART Plus

Ordering Information:

Model Name:

PICSTART Plus

Ordering Part Number:

DV003001

Devices Supported:

ALL

PIC12CXXX

PIC14X000

PIC16C5X

PIC16C6X

PIC16C7X

PIC16C8X

PIC16C9X

PIC17CXXX

System Description:

The PICSTART Plus development system includes the PICSTART Plus development programmer and the MPLAB IDE.

The PICSTART Plus programmer gives the product developer the ability to program user software into any of the supported microcontrollers. The PICSTART Plus software running under MPLAB provides for full interactive control over the programmer.

The MPASM macro assembler provides programmable memory data files, listing files and special files required for symbolic debug. The MPLAB-SIM software simulator allows the user to isolate code problems and debug firmware designs on PICmicro devices. It simulates the core functions as well as most of the peripherals of the PICmicro MCU families. It is particularly suitable for optimizing algorithms where real-time emulation is not required.

To order or obtain more information about PICSTART Plus or any other Microchip product, contact the Microchip sales office nearest you.

Customer Support:

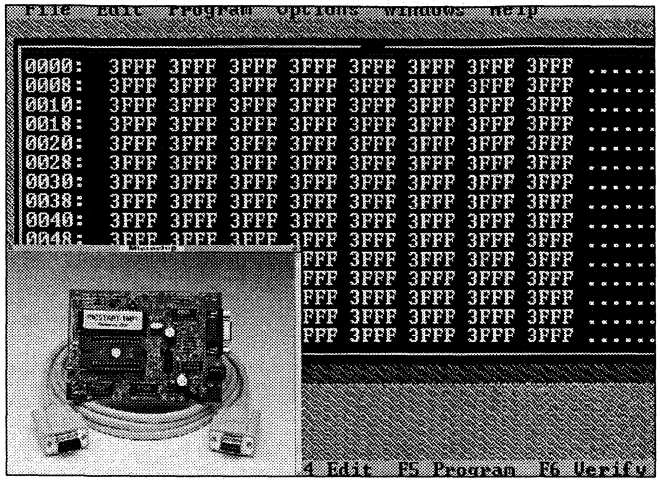
Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers and Corporate Application Engineers as well as a multifaceted Bulletin Board System. Microchip's Internet home page can be reached at: http://www.microchip.com

"The Microchip Advantage"	
MPLAB™	Integrated Development Environment (IDE)
MPASM	Universal PICmicro macro-assembler
MPLAB-SIM	Software Simulator
MPLAB-C	C compiler for PICmicro MCUs
PICMASTER®	Full-featured modular in-circuit emulator
ICEPIC	Low-cost modular in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART® Plus	Entry-level development kit with programmer
PICSTART® Lite	Low-cost entry level development kit
fuzzyTECH®-MP	Fuzzy Logic development software
MP-DriveWay™	Application Code Generator
KEELOQ®	Code Hopping Devices

© 1996 Microchip Technology Inc.

PICSTART®LITE

Ultra Low Cost Development Kit Product Brief



Features:

- Very low-cost introductory microcontroller design tool set
- DOS-based
- Operates with a PC-compatible host system
- Reads, programs and verifies EPROM Memories of PIC16C5X and PIC16CXXX microcontrollers
- PC Host Software provides file display, editing, and transfer device contents to and from programmer unit
- RS-232 interface cable
- Programmer unit connects to standard PC serial port
- MPASM Assembler for translation of assembler source code to object code for all PICmicro devices
- MPSIM, an MS-DOS based discrete-event simulator, is designed to model operation of all PIC16C5X and PIC16CXXX microcontrollers and debug
- PIC16C5X or PIC16CXXX device sample

PICSTART Lite makes designing with PIC16CXXX MCUs simple and affordable.

The PICSTART Lite development systems from Microchip Technology provide the product development engineer with a fast, easy and very low-cost way to begin evaluation and code development for base-line PIC16C5X and mid-range PIC16CXXX 8-bit one-time-programmable (OTP) microcontroller products where full real-time emulation is not required. PICSTART Lite is available in two versions, the PICSTART Lite-16B1 (28-pin DIP socket) and the PICSTART Lite-16C (40-pin DIP socket).

The PICSTART Lite development systems operate on any PC-compatible machine running the DOS operating system. Each development system includes an MS-DOS®-based software simulator (MPSIM), a macro assembler (MPASM), a microcontroller EPROM programmer board and a product sample. All supporting documentation and user's guides are supplied on 3-1/2" disks.

Sample software programs to run on the simulator are provided to help the development engineer quickly become familiar with the PICSTART Lite development system and PIC16C5X and PIC16CXX microcontroller families. A user need only provide his or her own preferred text editor and the system is ready for development of end products using one of Microchip's base-line or mid-range MCUs. PICSTART Lite is designed as a development programmer and is not recommended for use in a production environment.

11

© 1996 Microchip Technology Inc.

PICSTART Lite

Ordering Information:

Model Name:
PICSTART Lite-16B1

Ordering Part Number:
DV162003

Devices Supported:
PIC16C54, PIC16C54A, PIC16C55, PIC16C56, PIC16C57, PIC16C58A, PIC16C61, PIC16C62X, PIC16C71, PIC16C710, PIC16C711, PIC16C84

Model Name:
PICSTART Lite-16C

Ordering Part Number:
DV162002

Devices Supported:
PIC16C62, PIC16C62A, PIC16C63, PIC16C64, PIC16C64A, PIC16C65, PIC16C65A, PIC16C72, PIC16C73, PIC16C73A, PIC16C74

System Description:

The PICSTART Lite Development Programmer operates from a user supplied 9 volt power supply and connects with the supplied RS-232 (9 pin-D connector) cable to a serial port on the host PC. A full screen, user-friendly software program is provided for full interactive control over the programmer. Parts may be read, programmed, blank checked and verified. Configuration bits and ID locations may be specified.

The MPASM macro assembler provides translation of Assembler source code to object code with INHX8S, INHX8M and INHX32 output formats. The MPSIM discrete event software simulator models operation of PICmicro MCUs and allows the user to debug software that will use any of these devices. MPSIM features include program load and save, disassembler, symbolic debug, execution trace and breakpoints.

To order or obtain more information about PICSTART Lite or any other Microchip product, contact the Microchip sales office nearest you.

Customer Support:

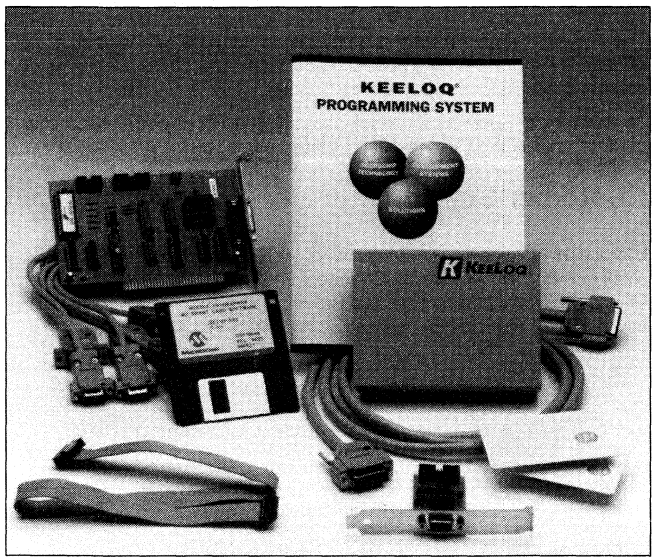
Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers and Corporate Application Engineers as well as a multifaceted Bulletin Board System. Microchip's Internet home page can be reached at: http://www.microchip.com

"The Microchip Advantage"	
MPLAB™	Integrated Development Environment (IDE)
MPASM	Universal PICmicro macro-assembler
MPLAB-SIM	Software Simulator
MPLAB-C	C compiler for PICmicro MCUs
PICMASTER®	Full-featured modular in-circuit emulator
ICEPIC	Low-cost modular in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART® Plus	Entry-level development kit with programmer
PICSTART® Lite	Low-cost entry level development kit
fuzzyTECH®-MP	Fuzzy Logic development software
MP-DriveWay™	Application Code Generator
KEELOQ®	Code Hopping Devices

© 1996 Microchip Technology Inc.

KEELOQ® Programming System

KEELOQ HCSXXX Device Programmers



Features:

- Set up HCSXXX encoder/decoder programming options
- Multiple algorithms for key generation
- Smartcard protection of manufacturer's keys
- Automatic serialization of encoders
- Logging to disk for factory record keeping
- Interface to automated test equipment
- Menu-driven user-friendly software includes:
  - Smartcard initialization program
  - Setup program to configure the various encoder/decoder programming options
  - Transmitter programming software
  - Receiver programming software
- Comprehensive User's Guide with easy-to-follow, step-by-step *Getting Started*

Programs KEELOQ Encoders and Decoders in manufacturing environment

The **KEELOQ® Programmer** provides manufacturing with the capability to program large quantities of HCS200, HCS300, HCS301, HCS360, HCS361 encoders and HCS509 and HCS512 decoders. The Code Hopping Encoder devices are used in the transmitter portion and the decoder devices in the receiver portion of secure RKE systems. The devices use the KEELOQ code hopping algorithm which combines high security, a small package outline and a very low cost to make this an ideal solution for unidirectional RKE systems.

Primary applications for the KEELOQ devices include automobile keyless entry and security systems, garage door openers, home security systems, central locking systems, gate openers, transaction cards, pre-paid telephone cards, vehicle immobilizers, identity tokens, electronic tagging, passive (batteryless) transponders, point-of-sale readers/adaptors and a growing list of other applications.

The KEELOQ Programmer is supplied with software to automate the programming of HCSXXX devices in the customer's factory production environment, and comes complete with accessories to be used with a host PC computer system. The KEELOQ Programmer includes an interface PCB, Smartcard reader, Smartcards and set of probes.

The versatility and security of the hardware-based key generation allow product range protection.

© 1996 Microchip Technology Inc.

KEELOQ Programming System

Ordering Information:

Model Name:

KEELOQ Programmer

Ordering Part Number:

PG306001

Devices Supported:

HCS200	Encoder
HCS300	Encoder
HCS301	Encoder
HCS360	Encoder
HCS361	Encoder
HCS509	Decoder
HCS512	Decoder

Host System Requirements:

- PC-compatible computer: 386DX, 486 or PENTIUM™ based with ISA or EISA bus
- EGA, VGA, 8514/A, Hercules graphic card (EGA or higher recommended)
- MS-DOS®/PC-DOS® v. 5.0 or greater
- Microsoft Windows 3.1 or greater in 386 enhanced mode
- 4 MB RAM, 4 MB free disk space
- 3.5 inch diskette drive
- One 8-bit PC/AT (ISA) I/O expansion slot (half size)

System Description:

The KEELOQ Programming System Hardware consists of a Key generation card used to securely generate encryption and decryption keys. The card may be plugged into a standard ISA bus slot on a PC. Also included is a Smartcard used to securely store the Manufacturer's Key, which is further protected by a PIN. A Smartcard reader and a set of spring-probes for connection to the transmitter PCB to be programmed complete the programming station hardware.

The accompanying user-friendly menu-driven Software, is supplied on 3.5" diskette and includes all the necessary software for programming large quantities of HCSXXX code hopping devices.

To order or obtain more information about KEELOQ or any other Microchip product, contact the Microchip sales office nearest you.

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers and Corporate Application Engineers as well as a multifaceted Bulletin Board System. Microchip's Internet home page can be reached at: http://www.microchip.com

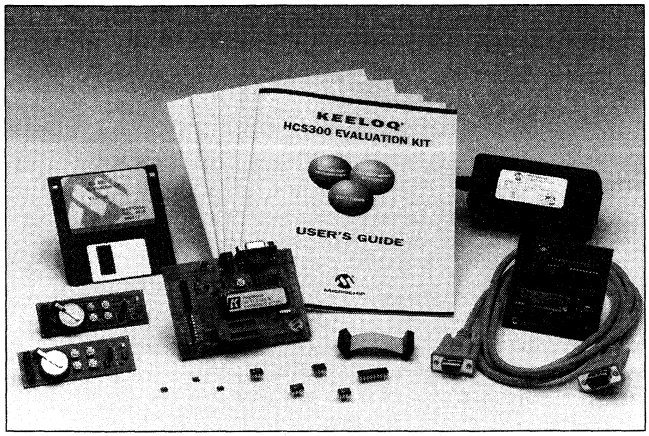
"The Microchip Advantage"	
MPLAB™	Integrated Development Environment (IDE)
MPASM	Universal PICmicro macro-assembler
MPLAB-SIM	Software Simulator
MPLAB-C	C compiler for PICmicro MCUs
PICMASTER®	Full-featured modular in-circuit emulator
ICEPIC	Low-cost modular in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART® Plus	Entry-level development kit with programmer
PICSTART® Lite	Low-cost entry level development kit
fuzzyTECH®-MP	Fuzzy Logic development software
MP-DriveWay™	Application Code Generator
KEELOQ®	Code Hopping Device Tools

© 1996 Microchip Technology Inc.

MICROCHIP

KEELOQ® Evaluation Kit

KEELOQ HCSXXX Evaluation Kit Product Brief



Features:

- Transmit hopping code messages using HCSXXX encoders
- Display the fixed and changing parts of hopping code messages on a PC
- Receive and decode hopping code transmissions using a PIC16C56 decoder and 93C46 EEPROM
- Learn new HCSXXX encoders onto the PIC16C56/93C46 decoder
- Program HCSXXX encoders
- Comprehensive User's Guide
- Schematics for transmitter and receiver

Let's you evaluate the capabilities of Microchip's code hopping devices

The KEELOQ Evaluation Kit demonstrates the capabilities of Microchip Technology's code hopping technology. The Code Hopping Encoder devices are designed to be the transmitter portion and the Decoder devices the receiver portion, of secure RKE* systems. The devices use the KEELOQ code hopping algorithm which combines high security, a small package outline and a very low cost to make this an ideal solution for unidirectional RKE systems.

Primary applications for the KEELOQ HCSXXX devices include automobile keyless entry and security systems, garage door openers, home security systems, central locking systems, gate openers, transaction cards, pre-paid telephone cards, vehicle immobilizers, identity tokens, electronic tagging, passive (batteryless) transponders, point-of-sale readers/adaptors and a growing list of other applications.

The KEELOQ Demonstration Kit includes all the necessary hardware to evaluate a code hopping system, as well as a basic demonstration software program, which is suppiied on a PC-compatible 3.5" diskette.

*Remote Keyless Entry

© 1996 Microchip Technology Inc.

KEELOQ Evaluation Kit

Ordering Information:

Model Name:

KEELOQ Evaluation Kit

Ordering Part Number:

DM303001

Devices Supported:

HCS300

HCS301

PIC16C56 Decoder

Host System Requirements:

- PC-compatible computer: 386DX, 486 or PENTIUM™ based with ISA or EISA bus
- Serial Port
- EGA, VGA, 8514/A, Hercules graphic card (EGA or higher recommended)
- MS-DOS®/PC-DOS® v. 5.0 or greater
- Microsoft Windows 3.1 or greater in 386 enhanced mode
- 4 MB RAM, 4 MB free disk space
- 3.5 inch diskette drive
- One 8-bit PC/AT (ISA) I/O expansion slot (half size)

System Description:

Hardware

- PIC16C56/93C46 EEPROM-based decoder
- RF Receiver module
- On-board +5V regulator and filtered rectifier for direct input from 9V AC/DC wall adapter
- Multiple radio frequency transmitters using HCS300 hopping code encoders
- RS232 socket and associated hardware for direct connection to RS232 interface
- Six bright LEDs connected to decoder outputs

Also includes all necessary software to program transmitters and to display codes or messages.

To order or obtain more information about KEELOQ or any other Microchip product, contact the Microchip sales office nearest you.

Customer Support:

Microchip maintains a worldwide network of distributors, representatives, local sales offices, Field Application Engineers and Corporate Application Engineers as well as a multifaceted Bulletin Board System. Microchip's Internet home page can be reached at: http://www.microchip.com

"The Microchip Advantage"	
MPLAB™	Integrated Development Environment (IDE)
MPASM	Universal PICmicro macro-assembler
MPLAB-SIM	Software Simulator
MPLAB-C	C compiler for PICmicro MCUs
PICMASTER®	Full-featured modular in-circuit emulator
ICEPIC	Low-cost modular in-circuit emulator
PRO MATE® II	Full-featured, modular device programmer
PICSTART® Plus	Entry-level development kit with programmer
PICSTART® Lite	Low-cost entry level development kit
fuzzyTECH®-MP	Fuzzy Logic development software
MP-DriveWay™	Application Code Generator
KEELOQ®	Code Hopping Device Tools

© 1996 Microchip Technology Inc.

PICDEM-1

Low-Cost PICmicro™ Demonstration Board

PRODUCT INFORMATION

The PICDEM-1 is a simple board which demonstrates the capabilities of several Microchip microcontrollers. The microcontrollers supported are: PIC16C5X (PIC16C54 to PIC16C58), PIC16C62X, PIC16C61, PIC16C62X, PIC16C71, PIC16C84, PIC17C42, PIC17C43 and PIC17C44. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5" disk. The users can program the samples (one each of PIC17C42, PIC16C71 and PIC16C55) provided with the PICDEM-1, on a PRO MATE®or PICSTART® programmer and easily debug/test the sample code, or the user can connect the PICDEM-1 with the PICMASTER™ emulator and download the sample code to the emulator and debug/test the code. Additionally, a generous 200-hole prototype area is available for the user to build some additional hardware and connect it to the microcontroller socket(s).

FEATURES:

Hardware:

- 40-pin, 28-pin and 18-pin Precision sockets for all supported microcontrollers.
- On board +5V regulator and filter rectifier for direct input from 9V AC/DC wall adapter.
- RS-232 socket and associated hardware for direct connection to RS-232 interface.
- 5K pot to simulate analog input for PIC16C71.
- Three push button Key for external stimulus and RESET.
- Eight bright LEDs connect to PORTB, help in displaying 8-bit binary values on PORTB.
- Socket for "canned" crystal Oscillator.
- Unpopulated holes provided for Xtal connection
- Jumper to disconnect on board RC Oscillator.
- 200-hole prototype area for user's hardware.



© 1995 Microchip Technology Inc.

Software:

- Program for PIC16C71 to demonstrate on-chip A/D features.
- Program for PIC16C84 to demonstrate on-chip EEPROM.
- Program for PIC17C42 to demonstrate on-chip USART.
- Program for PIC16C5X to demonstrate key input capability.
- All demo programs supplied on 3.5" disk,
- Additional programs available on Microchip's BBS.

DOCUMENTATION

- A comprehensive User's Guide with easy to follow step-by-step Getting Started and a Tutorial.
- Schematics for the entire circuit.

SAMPLES

Several UV erasable devices supplied are included. The device types may change from time to time. The supplied devices are typically:

- PIC17C42
- PIC16C71
- PIC16C55

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
DM163001	Low-cost Demonstration Board for PIC16C5X, PIC16C61, PIC16C62X, PIC16C71, PIC16C84, PIC17C42, PIC17C43 and PIC17C44.

© 1995 Microchip Technology Inc.

MICROCHIP

PICDEM-2

Low-Cost PIC16CXX Demonstration Board

PRODUCT INFORMATION

The PICDEM-2 is a simple board which demonstrates the capabilities of several Microchip microcontrollers, including PIC16C62, PIC16C63, PIC16C64, PIC16C65, PIC16C73 and the PIC16C74. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5" disk. A programmed sample is included, and the user may erase it and program it with the other sample programs using the PRO MATE® or PICSTART® programmer and easily debug and test the sample code. The PICDEM-2 is also usable with the PICMASTER® emulator, and all of the sample programs can be run and modified using the PICMASTER. Additionally, a generous prototype area is available for user hardware.

FEATURES:

Hardware:

- 40- and 28-pin DIP sockets
- On board +5V regulator for direct input from 9V AC/DC wall adapter or 9V battery.
- RS-232C socket and associated hardware for direct connection to RS-232C interface.
- 5K pot for analog inputs for the PIC16C73/74
- Three push button keys for external stimulus and RESET.
- Eight bright LEDs connected to PORTB for displaying 8-bit binary values.
- Socket for "canned" crystal oscillator.
- Unpopulated holes provided for crystal connection
- 128 x 8 Serial EEPROM.
- LCD module header.
- Keyboard header.
- Unpopulated holes for ACCESS.bus™ connector.



11

© 1995 Microchip Technology Inc.

Hardware (continued):

- Jacks for connection of 9V battery.
- Jumper to disconnect on-board RC oscillator.
- Prototype area for user hardware.

Software:

- Program for PIC16C74 to demonstrate on-chip A/D feature.
- Program for PIC16C64 to demonstrate I^2C™ Serial EEPROM usage.
- All demo programs supplied on 3.5" disk.
- Additional programs available on Microchip's BBS.

DOCUMENTATION:

- A comprehensive User's Guide with easy to follow, step-by-step Getting Started and Tutorial.
- Full schematics.

Samples:

Several UV erasable devices supplied are included. The device types may change from time to time. The supplied devices are typically:

- PIC16C64
- PIC16C74

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
DM163002	Low-cost Demonstration Board for PIC16C64, PIC16C65, PIC16C73 AND PIC17C74

© 1995 Microchip Technology Inc.

MICROCHIP

PICDEM-3

Low-Cost PIC16C9XX Demonstration Board

PRODUCT INFORMATION

The PICDEM-3 is a simple board which demonstrates the capabilities of Microchip's microcontroller LCD family. All necessary hardware is included to run basic demo programs, which are supplied on a 3.5" disk. A programmed sample is included, and the user may erase it and program it with the other sample programs using the PRO MATE® or PICSTART®Plus programmer. The PICSTART Plus requires a 68- to 40-pin converter socket.

The PICDEM-3 is also usable with the PICMASTER® emulator, and all of the sample programs can be run and modified using the PICMASTER. Additionally, a generous prototype area is available for user hardware.

All software is written in C using the MPLAB-C/demo version.

FEATURES:

Hardware:

- 68-pin PLCC sockets
- Unpopulated 44-pin PLCC socket for future 44-pin versions
- On board +5V regulator for direct input from 9V AC/DC wall adapter or 9V battery
- RS-232C socket and associated hardware for direct connection to RS-232C interface
- 5K pot for analog input to the PIC16C9XX
- Thermistor for use with A/D converter
- Three push button keys for external stimulus and RESET
- LCD Panel with 4 backplanes and 12 segments
- On-board LCD charge pump circuit with "unpopulated" external resistor ladder supported
- Socket for "canned" crystal oscillator
- Unpopulated holes provided for crystal or ceramic resonator connection
- Supports custom LCD Panel connection (up to 4 common, up to 28 segment) with common/segment header
- LCD pixel data converted to a digital result. Requires external hardware with SPI interface
- Serial port to communicate LCD pixel data to PC for display
- Keyboard header
- Jacks for connection of 9V battery
- Jumper to disconnect on-board RC oscillator
- Prototype area for user hardware

Software:

- MPLAB-C demo for PIC16CXX
- Sample programs which demonstrate:
  - real-time clock and display
  - temperature sensor and display
  - volt meter
  - USART using the SPI port
  - PC software to display LCD pixel information
- All demo programs supplied on 3.5" disk.
- Additional programs available on Microchip's BBS.

DOCUMENTATION:

- A comprehensive User's Guide with easy to follow, step-by-step Getting Started and Tutorial.
- Full schematics.

A UV erasable device is supplied. The device type may vary depending upon availability. The supplied devices are either:

- PIC16C924
- PIC16C923

SALES AND SUPPORT

To order or to obtain information, e.g., on the pricing or delivery, please use the listed part numbers, and refer to the listed sales offices.

PART NUMBER	DESCRIPTION
DM163003	Low-cost Demonstration Board for PIC16C923 and PIC16C924

11

© 1996 Microchip Technology Inc.

© 1996 Microchip Technology Inc.

SEEVAL®
DESIGNER'S KIT

Microchip Serial EEPROM Designer's Kit



FEATURES

- Includes everything necessary to begin developing Serial EEPROM-based applications
- Microchip *Total Endurance*™ software model
- Microchip *SEEVAL* evaluation and programming board
- Microchip *SEEVAL* software
- Microchip Serial EEPROM handbook
- Microchip Serial EEPROM sample pack
- RS-232 serial cable
- Power supply

SYSTEM REQUIREMENTS

- DOS 3.1 or higher
- Windows® 3.1
- VGA monitor
- 386 or 486 processor recommended
- Math coprocessor recommended

DEVICE SUPPORT

- Microchip 2-wire 24CXX/24LCXXB/85CXX
- Microchip *Smart Serial*™ 24XX65
- Microchip 3-wire 93CXX/93LCXX series
- Microchip 4-wire 59C11

11

© 1996 Microchip Technology Inc.

Designer's Kit

DESCRIPTION

Now designers of Serial EEPROM-based applications can enjoy the increased productivity, reduced time to market, and the ability to create a rock-solid design that only a well-thought-out development system can provide. Microchip's new Serial EEPROM Designer's Kit includes everything necessary to quickly develop a robust and reliable Serial EEPROM-based design and greatly reduce the time required for system integration and hardware/software debug.

The **Total Endurance software model** enables designers to quickly choose the best Serial EEPROM for the specific application and perform trade-off analysis with voltage, temperature, write cycle and other system parameters in order to achieve the desired Erase/Write endurance (specific ppm rate) or product lifetime. Total Endurance is the new standard of excellence in understanding and predicting the Erase/Write endurance of Serial EEPROMs. An on-line endurance tutorial is included, along with hypertext help files.

Microchip's **SEEVAL Serial EEPROM evaluation and programming system** will accept any Microchip Serial EEPROM in DIP package and enable the designer or system integrator to read, write, or erase any byte or the entire array. SEEVAL also provides the following advanced features to aid in system integration and debug:

- Program special user functions like *Smart Serial* configurations
- Hexadecimal display of array contents
- Pre-set or user-defined repeating patterns
- User-configurable functions like continuous read/write, programmable delay, etc.
- Upload/download files between the Serial EEPROM and disk

Another industry first, the **Microchip Serial EEPROM Handbook** provides a plethora of information crucial to the designers of Serial EEPROM-based systems. Along with data sheets on Microchip Serial EEPROMs, this resource provides application notes regarding Erase/Write endurance, interfacing with different protocols and many, many others. A cross-reference and selector guide are also included, plus article reprints and qualification reports on Microchip Serial EEPROMs.

USING SEEVAL AND TOTAL ENDURANCE

Both software packages can be loaded from Windows by choosing FILE RUN and entering SETUP.EXE from the Program Manager. The applications will install themselves; then a double mouse-click will start either application. The first step in either program is to select a device from the device list.

In Total Endurance, the user has simply to choose a Microchip Serial EEPROM device from the device-list menu and begin entering the application parameters. The entire process can take literally seconds to complete, and the model will output the PPM level and FIT rate of the device vs. the number of Erase/Write cycles. If the user has specified an application lifetime, the model will output PPM and FIT rates at that point in time. Alternately, the user may input a desired PPM level and the model will calculate the application lifetime which will result in that survival rate. The user may then trade-off any of the parameters (device type, voltage, application life, temperature, # of bytes per cycle, # of cycles per day etc.) to arrive at an optimal solution for the intended application.

Whenever a parameter is changed, calculation of the ppm/application life is automatic. An "update" box will appear inside the graph to indicate that new data has been entered and the graph should be redrawn. A single click in the "draw" box will redraw the plot of ppm vs. cycles; a click in the "Resize" box will take the plot to full-screen display for a closer view. The plot data can be saved to a file or the plot itself can be copied to the clipboard to be pasted into another application.

In **SEEVAL**, the user may choose to load a file from disk to program the Serial EEPROM, or read data from the EEPROM and save it to disk. The screen displays the contents of a software buffer. The buffer may be manipulated before programming data to the Serial EEPROM, or data can be written to the Serial EEPROM directly on-line. An area of memory can be highlighted (selected) and programmed with a predefined pattern or user-specified pattern. Alternately, the entire device can be programmed with any repeating pattern.

Both SEEVAL and Total Endurance allow the user to save any configuration as default. This configuration (device and application settings) will then automatically load at boot time.

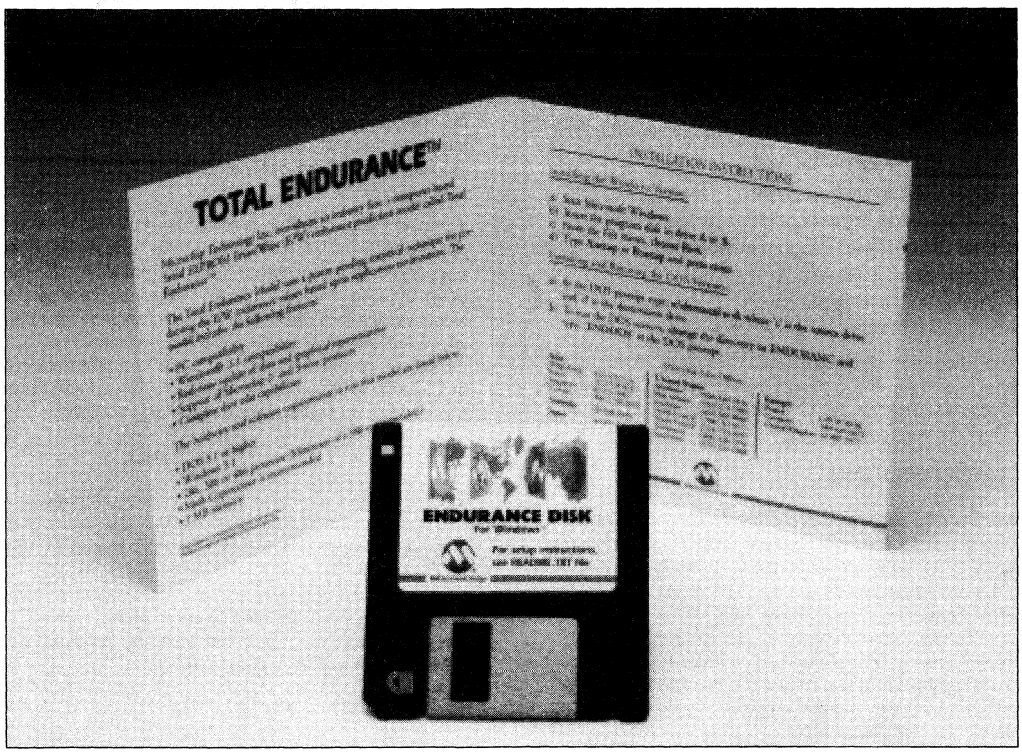
Order Information:

Description	Part Number
Serial EEPROM Designer's Kit	DV243001

© 1996 Microchip Technology Inc.

TOTAL ENDURANCE™

Microchip Serial EEPROM Endurance Model



FEATURES

- IBM® PC compatibility
- Windows® 3.1 or DOS 3.1 compatibility
- Automatic or manual recalculation
- Real-time update of data
- Full-screen or windowed graphical view
- Hypertext on-screen help
- Key or slide-bar entry of parameters
- On-screen editing of parameters
- Single-click copy of plot to clipboard
- Numeric export to delimited text file
- On-disk Endurance Tutorial

SYSTEM REQUIREMENTS

- DOS 3.1 or higher
- Windows 3.1
- 1MB memory
- 386 or 486 processor recommended
- Math coprocessor recommended

DEVICE SUPPORT

- Microchip 2-wire 24CXX/24LCXXB/24AAXX/ 85CXX
- Microchip 3-wire 93CXX/93LCXX/93AAXX Series
- Microchip 4-wire 59C11

11

© 1996 Microchip Technology Inc.

Total Endurance™

DESCRIPTION

Microchip's revolutionary Total Endurance Model provides electronic systems designers with unprecedented visibility into Serial EEPROM-based applications. This advanced software model (with a very friendly user interface) eliminates time and guesswork from Serial EEPROM-based designs by accurately predicting the device's performance and reliability within a user-defined application environment. Design trade-off analysis which formerly consumed days or weeks can now be performed in minutes...with a level of accuracy that delivers a truly robust design.

Users may input the following application parameters:

- Serial EEPROM device type
- Bytes to be written per cycle
- Cycling mode - byte or page
- Data pattern type - random or worst-case
- Temperature in °C
- Erase/Write cycles per day
- Application lifetime or target PPM level

The model will respond with FIT rate, PPM level, application life and a plot of the PPM level vs. number of cycles. The model is available in both DOS and Windows versions.

BACKGROUND

Microchip's research into the Erase/Write endurance of Serial EEPROMs has resulted in the conclusion that endurance depends upon three primary effects: the physical properties of the EEPROM cell, the internal error-correction technology employed, and the application environment. EEPROM endurance specified as a "typical" value in device data sheets must therefore be evaluated on a case-by-case basis, taking into account the manner in which the device will be used in the application. The Microchip Total Endurance™ software applies the user-defined application parameters to a complex mathematical model in order to emulate the EEPROM's performance and reliability in the system.

USING THE MODEL

The user has simply to choose a Microchip Serial EEPROM device from the device-list menu and begin entering the application parameters. The entire process can take literally seconds to complete, and the model will output the PPM level and FIT rate of the device vs. the number of Erase/Write cycles. If the user has specified an application lifetime, the model will output PPM and FIT rates at that point in time. Alternately, the user may input a desired PPM level and the model will calculate the application lifetime which will result in that survival rate. The user may then trade-off any of the parameters (device type, voltage, application life, temperature, # of bytes per cycle, # of cycles per day etc.) to arrive at an optimal solution for the intended application.

Whenever a parameter is changed, calculation of the ppm/application life is automatic. An "update" box will appear inside the graph to indicate that new data has been entered and the graph should be redrawn. A single click in the "draw" box will redraw the plot of ppm vs. cycles; a click in the "Resize" box will take the plot to full-screen display for a closer view. The plot data can be saved to a file or the plot itself can be copied to the clipboard to be pasted into another application.

ACCURACY OF THE MODEL

The accuracy of the Microchip Total Endurance model has been verified against test data to within ten percent of the actual values. However, Microchip makes no warranty as to its accuracy or applicability of the information to any given application. It is intended to be used as a guide to aid designers of Serial EEPROM-based systems in performing trade-off analysis and developing robust and reliable designs.

Order Information:

Description	Part Number
Total Endurance Software Disk	SW242001

© 1996 Microchip Technology Inc.

SECTION 12
OFFICE LOCATIONS

Factory Representatives 12-1
Distributors 12-7
Factory Sales 12-21

© 1997 Microchip Technology Inc.

© 1997 Microchip Technology Inc.

MICROCHIP

Factory Representatives

AFRICA

Tempe Technologies
8, 11th Avenue
Corner 7th Street
Houghton Estate
Johannesburg
Rep. of SA
Tel: 27-11-447-5186
Fax: 27-11-447-5959

ASIA

Goldenchip
11/F, 1116-7 Kwai on Fty Bldg.
103 Tai Lin Pai Road
Kwai Chung N.T.
Hong Kong
Tel: 852 2426-3968
Fax: 852 2481-7403

CANADA

Alberta

Enerlec Sales Ltd.
#103, 155 Glendeer Circle SE
Calgary, Alberta T2H 2S8
Canada
Tel: 403 777-1550
Fax: 403 777-1553

British Columbia

Enerlec Sales Ltd.
#7 3671 Viking Way
Richmond BC V6V 1W1
CANADA
Tel: 604 273-0882
Fax: 604 273-0884

Ontario

Pipe-Thompson Tech.
1394 Leigh Crescent
Gloucester ON K1J 8E4
Canada
Tel: 613 745-5927
Fax: 613 591-0461

Pipe-Thompson Tech.
3 Peary Way
Kanata ON K2L 7Z9
Canada
Tel: 613 591-1821
Fax: 613 591-0461

Pipe-Thompson Tech.
4 Robert Speck Pkway, Suite 1170
Mississauga ON L4Z 1S1
Canada
Tel: 905 281-8281
Fax: 905 281-8550

Quebec

Pipe-Thompson Tech.
4285 Rue Acres
Pierrefonds QC H9H 2T9
Canada
Tel: 514 624-8760
Fax: 514 624-0337

Pipe-Thompson Tech.
4193 Pinewood
Pierrefonds QC H9H 2W1
Canada
Tel: (514) 624-0595
Fax: (514) 624-0689

EUROPE

Austria

Active Rep GmbH
Wiessenstr. 40
A-4600 Wels
Austria
Tel: 43-7242-57-246-23
Fax: 43-7242-57-246-24

England

Arizona Technologies
Unit 21
Loughborough Technology Centre
Epinal Way
Loughborough
Leicestershire LE11 OQE
England
Tel: 44-1509-611277
Fax: 44-1509-611288

Germany

Active Rep GmbH
Vertrieb Nord
Emil Pleitner Gang 2
D-26135 Oldenburg
Germany
Tel: 49-441-20-699-86
Fax: 49-441-20-699-89

Active Rep GmbH
Vertrieb Nord
Obenitterstr. 21
D-42719 Solingen
Germany
Tel: 49-212-23-04-400
Fax: 49-212-23-04-023

Active Rep GmbH
Hugsburgestr. 43
D-82110 Germering
Germany
Tel: 49-89-89-42-96-60
Fax: 49-89-89-42-96-62

Active Rep GmbH
KennedyStr. 5
D-75438 Knittlingen
Germany
Tel: 49-7043-94-00-11
Fax: 49-7043-3-34-92

© 1997 Microchip Technology Inc.

Factory Representatives

Ireland

Eltech Agencies Ltd.
7 - 10 Robert Scott House
Patrick's Quay
Co. Cork
Ireland
Tel: 011-353-21-509-366
Fax: 011-353-21-509-344

Italy

Claitron Spa
V. Le Fulvio Testi
280/B
20126 Milano
Italy
Tel: 39-2-661491
Fax: 39-2-66105666

Scotland

Juniper Solutions
Enterprise House
Springkerse Business Park
Sterling FK7 7UF
Scotland
Tel: 011-31-1786-446220/1
Fax: 011-31-1786-446223

Switzerland

Mero
Microchip Engineering & Rep Org.
Switzerland/Liechtenstein
Heuerweg 1
CH-5605 Dottikon
Switzerland
Tel: 41-56-610-15-01
Fax: 41-56-610-15-03

MEXICO

CompTech Sales de Mexico
Av. Morelos Sur 882
Col. Centro
Cuernavaca, Mor. CP62000
Mexico
Tel: 011 52-7-318-3572
Fax: 011 52-7-318-5500

CompTech Sales
Prado de los Cedros
No. 221
Ciudad del Sol. Zapopan, Jalisco
Guadalajara CP45050
Mexico
Tel: 011 52-3-647-8137
Fax: 011 52-3-122-3303

CompTech Sales
Simona Barba 5100-9
Centro Ejecutivo "CELU"
Cd Juarez CHIH CP32330
Mexico
Tel: (915) 566-1022
Fax: 011 52-1-613-2156

UNITED STATES

Alaska

Micro Sales
2122-112th Avenue N.E.
Bellevue WA 98004-2947
Tel: 206 451-0568
Fax: 206 453-0092

Alabama

Concord Components
190 Lime Quarry Road
Suite 102
Madison AL 35758
Tel: 205 772-8883
Fax: 205 772-8262

Arizona

Western High Tech
9414 East San Salvador, Suite 206
Scottsdale AZ 85258
Tel: 602 860-2702
Fax: 602 860-2712

California

Escondido
Eagle Tech. Sales
1900 Sunset Drive, Suite A
Escondido CA 92025
Tel: 619 743-6550
Fax: 619 743-6585

Irvine
CTI Sales
16253 Laguna Canyon Road, Suite 160
Irvine CA 92718
Tel: 714 450-0170
Fax: 714 450-0838

Sunnyvale
Trinity Technologies
1261 Oakmead Pkway
Sunnyvale CA 94086
Tel: 408 733-9000
Fax: 408 733-9970

Colorado

Western Region Mktg.
9176 Marshall Place
Westminster CO 80030
Tel: 303 428-8088
Fax: 303 426-8585

Western Region Mktg.
6328 Brightstar Drive
Colorado Springs CO 80910
Tel: 719 548-8282
Fax: 719 548-0462

© 1997 Microchip Technology Inc.

Connecticut

Enfield
SJ New England
12 Gem Grove
Enfield CT 06082
Tel: 860 749-1531
Fax: Same as Phone

Guilford
SJ New England
10 Copper Ridge Circle
Guilford CT 06437
Tel: 203 458-7558
Fax: 203 458-1181

Naugatuck
SJ New England
15 Coventry Lane
Naugatuck CT 06770
Tel: 203 723-4707
Fax: 203 723-1629

Delaware

SJ Mid-Atlantic
131-D Gaither Drive
Mt. Laurel NJ 08054
Tel: 609 866-1234
Fax: 609 866-8627

District of Columbia

SJ Chesapeake
900 South Washington Street, Suite 307
Falls Church VA 22046
Tel: 703 533-2233
Fax: 703 533-2236

Florida

Altamonte Springs
Electramark Fla., Inc.
401 Whooping Loop, Suite 1565
Altamonte Springs FL 32701
Tel: 407 830-0844
Fax: 407 830-0847

Boca Raton
Electramark Fla., Inc.
215 North Federal Hwy, Suite 4E
Boca Raton FL 33432
Tel: 407 347-1717
Fax: 407 347-2997

Tampa
Electramark Fla., Inc.
2910 West Waters Avenue
Tampa FL 33614
USA
Tel: 813 915-1177
Fax: 813 915-1188

Georgia

Concord Components
6825 Jimmy Carter Blvd., Suite 1303
Norcross GA 30071
Tel: 770 416-9597
Fax: 770 441-0790

Hawaii

Trinity Technologies
1261 Oakmead Pkway
Sunnyvale CA 94086
Tel: 408 733-9000
Fax: 408 733-9970

Idaho

Micro Sales
1905 NW 169th Place, Suite D
Beaverton OR 97006
Tel: 503 645-2841
Fax: 503 645-3754

Northern
Micro Sales
2122-112th Avenue N.E.
Bellevue WA 98004-2947
Tel: 206 451-0568
Fax: 206 453-0092

Illinois

Northern
Janus, Inc.
650 East Devon Avenue, Suite 170
Itasca IL 60143
USA
Tel: 630 250-9650
Fax: 630 250-8761

Southern
Spectrum Sales
5494 Brown Road, Suite 124
St. Louis MO 63042
Tel: 314 731-4477
Fax: 314 731-1332

Indiana

Fort Wayne
Electro Reps
125 Airport North Office Park
Fort Wayne IN 46825
Tel: 219 489-8205
Fax: 219 489-8408

Indianapolis
Electro Reps
7240 Shadeland Station, Suite 275
Indianapolis IN 46256-3928
Tel: 317 842-7202
Fax: 317 841-0230

Sellersburg
Electro Reps
60 Buck Blvd.
Sellersburg IN 47172
Tel: (812) 246-2342
Fax: (812) 246-3286

Iowa

Spectrum Sales
708 J Avenue N.E., Suite 9
Cedar Rapids IA 52402
Tel: 319 366-0576
Fax: 319 366-0635

Kansas

Spectrum Sales
5382 West 95th Street
Prairie Village KS 66207
Tel: 913 648-6811
Fax: 913 648-6823

Kentucky

Northern
TMC
7775 Cooper Road, Suite 3
Cincinnati OH 45242
Tel: 513 984-6720
Fax: 513 984-6874

Southern
Electro Reps
60 Buck Blvd.
Sellersburg IN 47172
Tel: (812) 246-2342
Fax: (812) 246-3286

Louisiana

CompTech Sales
11191 Westheimer, Suite 890
Houston TX 77042
Tel: 713 781-7420
Fax: 713 890-1385

Maine

SJ New England
Corp. Place, Suite 3
267 Boston Road
North Billerica MA 01862
Tel: 508 670-8899
Fax: 508 670-8711

Maryland

SJ Chesapeake
900 South Washington Street, Suite 307
Falls Church VA 22046
Tel: 703 533-2233
Fax: 703 533-2236

Massachusetts

SJ New England
Corp. Place, Suite 3
267 Boston Road
North Billerica MA 01862
Tel: 508 670-8899
Fax: 508 670-8711

Michigan

Miltimore Sales, Inc.
22765 Heslip Drive
Novi MI 48375-4130
Tel: (810) 349-0260
Fax: (810) 349-0756

Minnesota

Com-Tek Sales
3502 Shoreline Drive
P. O. Box 17
Navarre MN 55392-0017
Tel: 612 471-7181
Fax: 612 471-7238

© 1997 Microchip Technology Inc.

Factory Representatives

Mississippi

Concord Components
190 Lime Quarry Road, Suite 102
Madison AL 35758
Tel: 205 772-8883
Fax: 205 772-8262

Missouri

Spectrum Sales
5382 West 95th Street
Prairie Village KS 66207
Tel: 913 648-6811
Fax: 913 648-6823

Spectrum Sales
5494 Brown Road, Suite 124
St. Louis MO 63042
Tel: 314 731-4477
Fax: 314 731-1332

Montana

Western Region Mktg.
9176 Marshall Place
Westminster CO 80030
Tel: 303 428-8088
Fax: 303 426-8585

Nebraska

Spectrum Sales
5382 West 95th Street
Prairie Village KS 66207
Tel: 913 648-6811
Fax: 913 648-6823

Nevada

Northern
Trinity Technologies
1261 Oakmead Pkway
Sunnyvale CA 94086
Tel: 408 733-9000
Fax: 408 733-9970

Southern
Western High Tech
9414 East San Salvador, Suite 206
Scottsdale AZ 85258
Tel: 602 860-2702
Fax: 602 860-2712

New Hampshire

SJ New England
Corp. Place, Suite 3
267 Boston Road
North Billerica MA 01862
Tel: 508 670-8899
Fax: 508 670-8711

New Jersey

Northern
Parallax
734 Walt Whitman Road, Suite 209
Melville NY 11747
Tel: 516 351-1000
Fax: 516 351-1606

Southern
SJ Mid-Atlantic
131-D Gaither Drive
Mt. Laurel NJ 08054
Tel: 609 866-1234
Fax: 609 866-8627

New Mexico

Western High Tech
9414 East San Salvador, Suite 206
Scottsdale AZ 85258
Tel: 602 860-2702
Fax: 602 860-2712

New York

Melville
Parallax
734 Walt Whitman Road, Suite 209
Melville NY 11747
Tel: 516 351-1000
Fax: 516 351-1606

Rochester
Apex Associates
1210 Jefferson Road
Rochester NY 14623
Tel: 716 272-7040
Fax: 716 272-7756

North Carolina

Zucker Associates
4070 Barret Drive
P. O. Box 19868
Raleigh NC 27619
Tel: 919 782-8433
Fax: 919 782-8476

North Dakota

Com-Tek Sales
3502 Shoreline Drive
P. O. Box 17
Navarre MN 55392-0017
Tel: 612 471-7181
Fax: 612 471-7238

Ohio

Cincinnati
TMC
7775 Cooper Road, Suite 3
Cincinnati OH 45242
Tel: 513 984-6720
Fax: 513 984-6874

Middleburg Heights
TMC
One Independence Place
4807 Rockside Road, Suite 360
Independence OH 44131
Tel: 216 520-0150
Fax: 216 520-0190

Oklahoma

CompTech Sales
18700 Woodbriar Lane
Catoosa OK 74015
Tel: 918 266-1966
Fax: 918 266-1808

Oregon

Micro Sales
1905 NW 169th Place, Suite D
Beaverton OR 97006
Tel: 503 645-2841
Fax: 503 645-3754

Pennsylvania

Eastern
SJ Mid-Atlantic
131-D Gaither Drive
Mt. Laurel NJ 08054
Tel: 609 866-1234
Fax: 609 866-8627

Western
TMC
20530 Route 19 North, Suite 8
Cranberry Township PA 16066
Tel: 412 779-2140
Fax: 412 779-4785

TMC
7775 Cooper Road, Suite 3
Cincinnati OH 45242
Tel: 513 984-6720
Fax: 513 984-6874

Rhode Island

SJ New England
Corp. Place, Suite 3
267 Boston Road
North Billerica MA 01862
Tel: 508 670-8899
Fax: 508 670-8711

South Carolina

Zucker Associates
4070 Barret Drive
P. O. Box 19868
Raleigh NC 27619
Tel: 919 782-8433
Fax: 919 782-8476

South Dakota

Com-Tek Sales
3502 Shoreline Drive
P. O. Box 17
Navarre MN 55392-0017
Tel: 612 471-7181
Fax: 612 471-7238

© 1997 Microchip Technology Inc.

Tennessee

Eastern

Concord Components
6825 Jimmy Carter Blvd., Suite 1303
Norcross GA 30071
Tel: 770 416-9597
Fax: 770 441-0790

Zucker Associates
4070 Barret Drive
P. O. Box 19868
Raleigh NC 27619
Tel: 919 782-8433
Fax: 919 782-8476

Western

Concord Components
190 Lime Quarry Road, Suite 102
Madison AL 35758
Tel: 205 772-8883
Fax: 205 772-8262

Texas

Austin

CompTech Sales
8760-A Research Blvd., Suite 493
Austin TX 78758
Tel: 512 272-9465
Fax: 512 272-9436

Brownsville

CompTech Sales
2390 Central Blvd., Suite P
Brownsville TX 78520
Tel: 210 504-9693
Fax: 210 504-9982

Houston

CompTech Sales
11191 Westheimer, Suite 890
Houston TX 77042
Tel: 713 781-7420
Fax: 713 890-1385

Irving

CompTech Sales
2401 Gateway Drive, Suite 114
Irving TX 75063
Tel: (972) 751-5500
Fax: (972) 550-8113

Utah

Western Region Mktg.
3539 South Main, Suite 210
Salt Lake City UT 84115
Tel: 801 268-9768
Fax: 801 268-9796

Vermont

SJ New England
Corp. Place, Suite 3
267 Boston Road
North Billerica MA 01862
Tel: 508 670-8899
Fax: 508 670-8711

Washington

Micro Sales
2122-112th Avenue N.E.
Bellevue WA 98004-2947
Tel: 206 451-0568
Fax: 206 453-0092

West Virginia

TMC
20530 Route 19 North, Suite 8
Cranberry Township PA 16066
Tel: 412 779-2140
Fax: 412 779-4785

Wisconsin

Janus, Inc.
375 Williamstowne
Delafield WI 53018
Tel: 414 646-5420
Fax: 414 646-5421

Com-Tek Sales
3502 Shoreline Drive
P. O. Box 17
Navarre MN 55392-0017
Tel: 612 471-7181
Fax: 612 471-7238

Wyoming

Western Region Mktg.
9176 Marshall Place
Westminster CO 80030
Tel: 303 428-8088
Fax: 303 426-8585

© 1997 Microchip Technology Inc.

Factory Representatives

© 1997 Microchip Technology Inc.

MICROCHIP

Distributors

AFRICA

South Africa

Pace Electronic Components (Pty) Ltd.
Cnr. Vanacht & Gewel Streets
P.O. Box 701
Isando 1600, Transvaal
RSA
Tel: 27-11-9741211/6
Fax: 27-11-9741271

ASIA/PACIFIC

Australia

Avnet Pacific
Unit C, 6-8 Lyon Park Rd.
North Hyde NSW 2113
Australia
Tel: 612-9878-1299
Fax: 612-9878-1299

Zatek Australia Pty. Ltd.
9-10 Bastow Place
P. O. Box 613
Mulgrave, Victoria 3170
Melbourne
Australia
Tel: 613-95749644
Fax: 613-95749661

Zatek Australia Pty. Ltd.
Level 4,5
Belmore Street
Burwood N.S.W.
2134 Sydney
Australia
Tel: 61-2-744-5711
Fax: 61-2-744-5527

China

Excelpoint Systems Ltd.
79 JIA, Fuxing Road
Haidian District
Beijing 100854 PRC
Tel: 86-10-6837-3894
Fax: 86-10-6838-5621

Excelpoint Systems Ltd.
Rm. 610, Mingmao Bldg.
Shuncheng Road
Chengdu 610015 PRC
Tel: 86-28-662-8872
Fax: 86-28-662-8872

Excelpoint Systems Ltd.
Rm. A1108,
Yinhai Commercial Bldg.
250 Caoxi Road
Shanghai 200233 PRC
Tel: 86-21-6482-2280
Fax: 86-21-6482-4680

Fuzhou Dingxu
3F/4F, 3rd Bldg.,
Chang Ting Industrial Area, Xianjin Road
Fuzhou 350004 PRC
Tel: 86-591-372-9174
Fax: 86-591-371-7704

Goldenchip Elect. Tech Co. Ltd.
7 Floor, 275 Wusi Road
Fuzhou 350003 PRC
Tel: 86-591-771-1118
Fax: 86-591-771-4160

Goldenchip Research
8 Floor, Hua Jian Bldg.
115 Wuyi Middle Road
Fuzhou 350005 PRC
Tel: 86-591-331-3244
Fax: 86-591-331-3244

Infinitron/SI Logic Ltd.
Block B, 1/fl.
153 FuRongJiang Road
Shanghai 200335 PRC
Tel: 86-21-62330879
Fax: 86-21-62330878

Weikeng Ind. Co. Ltd.
Rm. 1201, Unit C
Hui Yuan International Apartment
8 An Li Rd., An Ding Men Wai
Beijing 100101 PRC
Tel: 86-10-6492-3726
Fax: 86-10-6499-1424

Weikeng Ind. Co. Ltd.
18 Floor, Unit G, Mira Garden
1720 HongQiao Road
Shanghai 200335 PRC
Tel: 86-21-6270-5509
Fax: 86-21-6270-5510

Wuhan Liyuan Research
15 Zuo Dao Quan Road
Wuhan 430070 PRC
Tel: 86-27-749-3500
Fax: 86-27-749-3497

© 1997 Microchip Technology Inc.

Distributors

ASIA/PACIFIC (Continued)

Hong Kong

Excelpoint Systems Ltd.
Rm. 1506, 15/F Telford House
16 Wang Hoi Road
Kowloon Bay
Hong Kong
Tel: 852-2503-2212
Fax: 852-2503-1558

Infinitron Asia Ltd.
802, Kinox Centre
9 Hung To Road
Kwun Tong
Kowloon
Hong Kong
Tel: 852-2341-6611
Fax: 852-2950-0987

Weikeng Ind. Co. Ltd.
1509, Chevalier Comm Centre
Wang Hoi Road
Kowloon Bay
Kowloon
Hong Kong
Tel: 852-27999035
Fax: 852-279966968

India

Excelpoint Systems India
3318 7th Cross
13th Main, HAL 2nd Stage
Bangalore 560 008
India
Tel: 91-80-5263148
Fax: 91-80-5261102

Primetech Communications
No. 289, 5th Cross,
1st Stage, 1st Phase
Gokul, Bangalore 560 054
India
Tel: 91-80-337-8553
Fax: 91-80-559-9840

Japan

Dianichi Contronics Inc.
Dainichi Bldg. 1-1-7 Koraku
Bunkyo-Ku
Tokyo 112
Japan
Tel: 81-3-3818-8081
Fax: 81-3-3818-8088

Global Electronics Corp.
Nichibei Time 24 Bldg.
35 Tansu-Cho, Shinjuku-Ku
Tokyo 162
Japan
Tel: 81-3-3260-1411
Fax: 81-3-3260-7100

Marubeni Hytech Co.
Nissei Iidabashi Bldge
1-18 Ageba-cho, Shinjuku-ku
Tokyo 162
Japan
Tel: 81-3-5228-7241
Fax: 81-3-5228-7249

Ryoden Trading Co.
3-15-15, Higashi Ikebukuro
Toshima-Ku
Tokyo 170
Japan
Tel: 81-3-5396-6211
Fax: 81-3-5396-6443

Unidux, Inc.
5-1-21, Kyonan-cho
Musashino-shi
Tokyo 180
Japan
Tel: 81-422-31-4111
Fax: 81-422-31-2050

Korea

Hanil System
Nam-Kyong Bldg. 4F
418-4 Dogok 2-Dong
Kangnam-gu
Seoul 135-272
Korea
Tel: 822-579-3194
Fax: 822-579-1208

ILJI Telecom Co., Ltd.
J-AD B/D, 4th Floor
#121-7, Yang Jae-Dong
Seocho-Gu
Seoul 137-130
Korea
Tel: 82-2-573-6800
Fax: 82-2-571-3936

M & K Sang SA
Room 516, Dong-A Bldg.
14-2, Yoido-Dong
Youngdeungpo-Ku
Seoul 150-010
Korea
Tel: 822-785-1127
Fax: 822-785-1129

Mi. Tech Corp
Kyumsan B/D, 2F
629-3, Yeoksam-Dong
Kangnam-ku
Seoul 135-080
Korea
Tel: 822-563-0301
Fax: 822-565-2194

Nasco Co., Ltd.
23rd Floor, Hanwha Securities Bldg.
23-5, Yoido-Dong, Youngdeungpo-Gu
Seoul
Korea
Tel: 82-2-3772-6820
Fax: 82-2-3772-6801

Prochips Inc.
779-12, Daelim 3-Dong
Youngdeungpo-Ku,
Seoul 150 073
Korea
Tel: 822-849-8567
Fax: 822-849-8659

New Zealand

Avnet Pacific
274 Church Street
Private Bag 92821
Penrose Auckland
New Zealand
Tel: 649-636-7801
Fax: 649-634-4900

Avnet Pacific
P. O. Box 30769
Lower Hutt
New Zealand
Tel: 644-527-3023
Fax: 644-527-3024

Singapore

Excelpoint Systems Ltd.
24 Kaki Bukit Crescent
Kaki Bukit TechPark 1
416255
Singapore
Tel: 65-741-8966
Fax: 65-741-8980

Gates Engineering Pte. Ltd.
1123 Serangoon Road
#03-01 UMW Building
328207
Singapore
Tel: 65-299-9937
Fax: 65-299-7636

Parallax Research
9 International Business Park, Unit 02-05
609915
Singapore
Tel: 65-665-6938
Fax: 65-665-1369

© 1997 Microchip Technology Inc.

ASIA/PACIFIC (Continued)

Taiwan

A-Mean, Inc.
No. 2-2, Ally 1, 220 Lane,
Jing-An Road, Chong Ho City
Taipei Shian
Taiwan ROC
Tel: 886-2-2453779
Fax: 886-2-2480484

Bright Systems Co., Ltd.
6F-2, # 160
Ming Chuan E. Rd., Sec. 6
Taipei Taiwan R.O.C
Tel: 886-2-791-5766
Fax: 886-2-794-9070

GoldenTechnology Corp.
4F, # 221, Chung Yang Road
Nan Kang Dist.
Taipei Taiwan ROC
Tel: 886-2-652-2255
Fax: 886-2-651-0589

Infortech Co., Ltd.
6F-2, # 436, Fu Hsing N. Road
Taipei
Taiwan ROC
Tel: 886-2-509-2734
Fax: 886-2-509-2736

Letronic Technology Corp.
2F, # 189, Sec. 2
Chien-Kuo S. Road
Taipei Taiwan ROC
Tel: 886-2-709-3000
Fax: 886-2-709-1000

Microtime Computer Inc.
10FL, No. 196, Section 4,
Cheng-Teh Road
Taipei
Taiwan ROC
Tel: 886-2-8811791
Fax: 886-2-882-0836

Pinnacle Technologies Co. Ltd.
4F, # 270, Sec. 3
Nan Kang Road
Taipei Taiwan ROC
Tel: 886-2-7884800
Fax: 886-2-788-5969

Solomon Technology Corp.
7F, #2, Lane 47, Sec. 3
Nan Kang Road
Taipei
Taiwan
Tel: 886-2-788-8989
Fax: 886-2-788-8029

Weikeng Ind. Co. Ltd.
5F, # 34, Sec.1
Huan Shan Road, Nei Hu
Taipei Taiwan ROC
Tel: 886-2-695-0202
Fax: 886-2-685-0959

Thailand

Electronics Source Co., Ltd.
5 Banmoh Road
Wangburapapirom Pranakorn
Bangkok 10200
Thailand
Tel: 662-884-9210
Fax: 662-884-9213

CANADA

Alberta

Anthem Electronics Canada Ltd.
3015-5th Avenue
Suite 210
Calgary AB T2A 6T8
Canada
Tel: 403 273-2780
Fax: 403 273-7458

Future Electronics
2015 32nd Avenue N.E., Unit 1
Calgary AB T2E 6Z3
Canada
Tel: 403 250-5550
Fax: 403 291-7054

Future Electronics
6029-103rd Street N.E.
Edmonton AB T6H 2H3
Canada
Tel: 403 438-5888
Fax: 403 436-1874

British Columbia

Anthem Electronics Canada Ltd.
Imperial Square Lake City
8535 Commerce Ct.
Burnaby BC V5A 4N4
Canada
Tel: 604 606-8950
Fax: 604 421-0582

Future Electronics
3689 East 1st Avenue
Suite 200
Vancouver BC V5M 1C2
Canada
Tel: 604 294-1166
Fax: 604 294-1206

Pioneer-Standard Electronics
170-10711 Combie Road
Richmond BC V6E 2V3
Canada
Tel: 604 273-5575
Fax: 604 273-2413

Manitoba

Anthem Electronics Canada Ltd.
42 Moberly Avenue
Winnipeg MB R2C 4C1
Canada
Tel: 204 222-7281
Fax: 204 222-7287

Future Electronics
106 King Edward
Winnipeg MB R3H 0N8
Canada
Tel: 204 944-1446
Fax: 204 783-8133

© 1997 Microchip Technology Inc.

Distributors

Ontario

Anthem Electronics Canada Ltd.
300 North Rivermede Road
Concord ON L4K 2Z4
Canada
Tel: 905 738-1071
Fax: 905 738-4183

Anthem Electronics Canada Ltd.
39 Robertson Road. Suite 506
Bell Mews
Nepean ON K2H 8R2
Canada
Tel: 613 596-6980
Fax: 613 596-6987

Future Electronics
5935 Airport Road, Suite 200
Mississauga ON L4V 1W5
Canada
Tel: 416 612-9200
Fax: 416 612-9185

Future Electronics
1050 Baxter Road
Baxter Centre
Ottawa ON K2C 3P2
Canada
Tel: 613 820-8313
Fax: 613 820-3271

Future Electronics Corp.
237 Hymus Boulevard
Pointe Claire QC H9R 5C7
Canada
Tel: 514 694-7710
Fax: 514 695-3707

Pioneer-Standard Electronics
148 York Street
Suite 209
London ON N6A 1A9
Canada
Tel: 519 672-4666
Fax: 519 672-3528

Pioneer-Standard Electronics
3415 American Drive
Mississauga ON L4V 1T4
Canada
Tel: 905 405-8300
Fax: 905 405-6423

Pioneer-Standard Electronics
223 Colonnade Road
Suite 112
Nepean ON K2E 7K3
Canada
Tel: 613 226-8840
Fax: 613 226-6352

Quebec

Anthem Electronics Canada Ltd.
6600 Trans Canada Hwy
Suite 620
Pointe Claire QC H9R 4S2
Canada
Tel: 514 697-8149
Fax: 514 697-1210

Pioneer-Standard Electronics
2954 Blvd. Laurier
Suite 100
Ste-Foy QC G1V 4T2
Canada
Tel: 418 654-1077
Fax: 418 654-2958

Pioneer-Standard Electronics
520 McCaffrey Street
Ville St. Laurent, QC H4T 1N1
Canada
Tel: 514 737-9700
Fax: 514 737-5212

EUROPE

Austria

Avnet E2000 GmbH
Waidhausenstrasse 19
A-1140 Wien
Austria
Tel: 43-1911-28-47
Fax: 43-1911-38-53

Elbatex GmbH
Eitnergasse 6
A-1230 Wien
Austria
Tel: 43-1-86-642
Fax: 43-1-86-642

Belgium

Acal Betea
Lozenberg 4
1932 Zaventem
Belgium
Tel: 32-2-720-59-83
Fax: 32-2-725-10-14

Sonetech Nederland BV
Office Belgium
Limburg Stirum 243, B-2
B-1780 Wemmel
Belgium
Tel: 32-246-00707
Fax: 32-246-01200

Bulgaria

Elbatex Bul D.O.O.
1 Phillip Kutev Str.
BG-1474 Sofia
Bulgaria
Tel: 359-2-624-628
Fax: 359-2-962-5252

Croatia/Slovenia

Elbatex Bul D.O.O.
GF-Milan Susa 11
Stegne 25
SLO-61000 Laibach
Slovenia
Tel: 38-661-159-7198
Fax: 38-661-159-2398

Czech Republic

Elbatex Praha S.R.O.
Novodvorsku 994
CZ-14221 Praha
Czech Republic
Tel: 42-24-76-3707
Fax: 42-24-76-3708

© 1997 Microchip Technology Inc.

EUROPE (Continued)

Denmark

Arrow Exatec AS
Mileparken 20E
DK2740 Skovlund
Denmark
Tel: 45-44-927000
Fax: 45-44-926020

England

Arrow-Jermyn
St. Martins Business Centre
Cambridge Road
Bedford MK42 0LF
England
Tel: 44-1234-270027
Fax: 44-1234-791549

Electronic Services Ltd.
Edinburgh Way
Harlow
Essex CM20 2DF
England
Tel: 011-44-1279-441144
Fax: 011-44-1279-443417

Eurodis HB Electronics Ltd.
Lever Street
Bolton
Lancashire BL3 6BJ
England
Tel: 011-44-1204-555000
Fax: 011-44-1204-384911

Future Electronics Ltd.
Future House
Poyle Road, Colnbrook
Berks SL3 0EZ
England
Tel: 011-44-1753-763000
Fax: 011-44-1753-689100

Hawke Components Ltd.
26 Campbell Court
Bramley NR.Bassingstoke
Hants RG26 5EG
England
Tel: 44 1258-880800
Fax: 44 1256-880325

Finland

Arrow Field Oy
Niittylantie 5
Helsinki FIN-00620
Finland
Tel: 011-358-9-777-571
Fax: 011-358-9-798-853

Memec Finland Oy
Asemankello
Vernissankatu 6
Fin 01300 Vantaa
Finland
Tel: 358-9-836-2600
Fax: 358-9-8362-6027

France

Future France
Parc Technopolis LP 854
Les Ulis
3 Av. du Canada - Bat. Theta 2
Courtaboeuf 91974
France
Tel: 33-1-69-82-11-11
Fax: 33-1-69-82-11-00

Mecodis
Parc d'Activites
3 Allee des Erables
94042 Creteil
Cedex
France
Tel: 33-1-43-99-44-00
Fax: 33-1-43-99-98-28

MULTIcomposants
BP 69 - Saint Aubin
91192 Gif Sur Yvette
France
Tel: 33-1-69-85-83-00
Fax: 33-1-69-85-83-99

Germany

Avnet E2000 GmbH
Stahlgruberring 12
D-81829 Müchen
Germany
Tel: 49-89-4-5110-142
Fax: 49-89-4-5110-210

Avnet/Setron Schiffer-Elektronik GmbH
Friedrich-Seele-Str. 3a
D-38032 Braunschweig
Germany
Tel: 49-531-8098-130
Fax: 49-531-8098-205

Future Electronics Deutschland GmbH
Munchner Strasse 18
D-85774 Unterföhring
Germany
Tel: 49- 89-95727-0
Fax: 49- 89-95727-140

Metronik GmbH
Leonhardsweg 2
D-82008 Unterhaching
Germany
Tel: 49-89-61108-0
Fax: 49-89-61108-110

Rutronik RSC-Halbleiter GmbH
Industriestrasse 2
D-75228 Ispringen/Pforzheim
Germany
Tel: 49- 7231-8010
Fax: 49- 7231-82282

Semitron W. Rock GmbH
Im Gut 1
D-79790 Küssaberg
Germany
Tel: 49-7742-8001-0
Fax: 49-7742-6901

Greece

P. Caritato & Associates SA
Ilia Iliou 31
Athens 11743
Greece
Tel: 30-1-9020115
Fax: 30-1-9017024

© 1997 Microchip Technology Inc.

Distributors

EUROPE (Continued)

Hungary

Chipcad Electronics Distribution, Ltd.
Angol u. 42
H-1149 Budapest
Hungary
Tel: 36-1-363-6676
Fax: 36-1-363-7889

Elbatex HUN Kft.
GF-NN
Szigetvari U. 5
H-1083 Budapest
Hungary
Tel: 36-1-269-9093
Fax: 36-1-269-9096

Ireland

Top Tech Solutions Ltd.
124 Orby Drive
Belfast BT5 6BB
No. Ireland
Tel: 011-44-1232-704985
Fax: 011-44-1232-792563

Israel

Elina Electronics Ltd.
14 Raoul Wallenberg St.
P.O. Box 13190
Tel: Aviv 61131
Israel
Tel: 972-3-649-85-43
Fax: 972-3-649-87-45

Italy

Claitron Spa
Via Comune Antico, 43
20125 Milano
Italy
Tel: 39-2-66100971
Fax: 39-2-661-04279

Farnell Electronic Services
Viale Milanofiori, E/5
20094 Assago
Milan
Italy
Tel: 39-2-824701
Fax: 39-2-8242631

Kevin SRL
Via Venezia Giulia, 10
20157 Milano
Italy
Tel: 39-2-33200914
Fax: 39-2-33200917

Latvia

Ormix Ltd.
Kr. Barona 136
LV-1012
Riga
Latvia
Tel: 011-371-2292839
Fax: 011-371-2292823

Netherlands

Acal Auriema
Beatrix de Rijkweg 8
5657 EG Eindhoven
Netherlands
Tel: 31-40-2-502-602
Fax: 31-40-2-510-255

Sonetech Benelux
SEI Building
Takkebijsters 2, NL-4817 BI Breda
P.O. Bo 6824
NL-4802 HV Breda
Netherlands
Tel: 31-76-572-2333
Fax: 31-76-572-2395

Norway

Berendsen Components AS
P.O. Box 9376 Gronlund
N0135
Oslo
Norway
Tel: 011-47-22-088500
Fax: 011-47-22-088590

Component-74 Eidsvold
P.O. Box 9
N-2070 Raholt
Norway
Tel: 47-63-956010
Fax: 47-63-956019

Poland

Elbatex POL S.Z.O.
ul. Wilcza 50/52
PL-00-679 Warsaw
Poland
Tel: 48-22-621-71-22
Fax: 48-22-623-0605

Gamma Poland
ul. Sady Zoliborskie 13A
PL-01-772 Warsaw
Poland
Tel: 48-226-638376
Fax: 48-226-639887

Portugal

Digicontrole
Dpt Comercial
Av. Eng. Arantes e Oliveira 5-2 D
Olaias
1900 Lisboa
Portugal
Tel: 351-1-840-57-30
Fax: 351-1-849-03-73

Russia

Gamma Ltd.
111 Grazdanski Pr-T
St. Petersburg 197137
Russia
Tel: 7-812-5311-402
Fax: 7-812-5311-402

Slovakia

Elbatex SK S.R.O.
Kasmirska 7
SK-821 04 Bratislava
Slovakia
Tel: 42-75-72-4173
Fax: 42-75-22-0600

Spain

Sagitron
Corazon de Maria 80/82
28002 Madrid
Spain
Tel: 34-1-416-92-61
Fax: 34-1-413-5848

Sweden

Memec Scandinavia AB
Sehlstedsgatan 6
115 28 Stockholm
Sweden
Tel: 011-46-8-459-79-00
Fax: 011-46-8-459-79-99

Switzerland

Avnet E2000 AG
Bohnirainstrasse 11
Postfach 1575
CH-8801 Thalwil
Switzerland
Tel: 41-1722-13-30
Fax: 41-1722-13-40

Elbatex Gruppe AG
Hardstr. 72
CH-5430 Wettingen
Switzerland
Tel: 41-56-4375111
Fax: 41-56-4375454

Semitron W. Roeck & Co.
Promenadenstr.6
CH-5330 Zurzach
Switzerland
Tel: 41-56-2696040
Fax: 41-56-2493569

Turkey

Inter Muehendislik
Ve Ticaret A.S. 1
Hasircibasi Caddesi No. 55
81310 Kadikoy
Istanbul
Turkey
Tel: 90-216-349-9400
Fax: 90-216-1-349-9431

Ukraine

Vector
15 Amerovsky Side Street
Kharkov 310003
Ukraine
Tel: 011-7-572-431680
Fax: 011-7-572-473048

© 1997 Microchip Technology Inc.

MEXICO

Chihuahua
Dicopel S.A. de C.V.
Ave. 20 de Noviembre
No. 102 Desp. 1 Col. Centro
Chihuahua, Chih. CP 31000
Mexico
Tel: 52-14-11-27-07
Fax: 52-14-11-27-67

Guadalajara
Dicopel S.A. de C.V.
Av. Federalisimo Sur 368
Sector Juarez
Guadalajara, Jalisaco CP 44100
Mexico
Tel: 523-626-1232
Fax: 523-626-3966

Electronica Seta, S.A. de C.V.
Santa Mario No. 100
Col. Capalita
Zapopan, Jalisco CP45000
Mexico
Tel: 523-121-0338
Fax: 523-121-0338

Future Electronics de Mexico, S.A. de C.V.
Prol. Americas 1600 2do.
Guadalajara, Jalisco 44610
Mexico
Mexico
Tel: 913 678-9281
Fax: 913 678-9271

Future Electronics de Mexico, S.A. de C.V.
Chimalhuacan 3569
6o. Piso, Suite 2, Ciudad Del Sol
Zapopan, Jalisco CP45050
Mexico
Tel: 523-122-0043
Fax: 523-122-1066

La Loma
Electronica Seta, S.A. de C.V.
Galena 114, 2o. Piso
La Loma Tlalnepantla
Edo. De Mexico CP 54060
Mexico
Tel: 525-390-7713
Fax: 525-390-9468

Leon
Dicopel S.A. de C.V.
Blvd. Adolfo Lopez Mateos 340
Orienta Col. Centro
Leon, Gto. CP 37530
Mexico
Tel: 14-44-43
Fax:

Merida
Dicopel S.A. de C.V.
Av. Aviacion 602-A
(Al Lado De La Dina)
Merida, Yuc. CP 97000
Mexico
Tel: 529-984-0267
Fax: 529-0220

Mexico, D.F.
Dicopel S.A. de C.V.
Fco. PimenTel: 98
Col. San Rafael
Mexico D.F. CP 06470
Mexico
Tel: 52-57-03-18-19
Fax: 52-57-03-17-72

Monterrey
Dicopel S.A. de C.V.
Av. H Colegio Miltar 412
Col. del Prado
Monterrey, N.L. CP 64410
Mexico
Tel: 528-374-6693
Fax: 528-374-1944

SOUTH AMERICA

All Other Countries Except Brazil

Ibars Electronics Corp
10020 N.W. 6th CT
Pembroke Pines FL 33024
USA
Tel: 954 430-3740
Fax: 954 430-3763

Brazil

Artimar, Ltda.
Rua Marques De Itu
70 - 10 Andar
01223 Sao Paulo
Brazil
Tel: 011 55-11-231-0277
Fax: 011 55-11-255-0511

© 1997 Microchip Technology Inc.

Distributors

UNITED STATES

Alabama

Anthem Electronics, Inc.
4920 - H Corporate Drive
Huntsville AL 35805
Ten 205 890-0302
Fax: 205 890-0130

Bell Industries
555 Sparkman Drive, Suite 600D
Huntsville AL 35816
USA
Tel: 205 430-3150
Fax: 205 430-3160

Future Electronics
6767 Old Madison Place, Suite 400A
Huntsville AL 35806
Tel: 205 971-2010
Fax: 205 922-0004

Pioneer-Standard Electronics
4835 University Square
Suite 5
Huntsville AL 35816
Tel: 205 837-9300
Fax: 205 837-9358

Arizona

Anthem Electronics, Inc.
1406 West 14th Street, Suite 101
Tempe AZ 85281
Tel: 602 966-6600
Fax: 602 966-4826

Bell Industries
10611 North Hayden Road
Building D, Suite 103
Scottsdale AZ 85260
USA
Tel: 602 905-2355
Fax: 602 905-2356

Future Electronics
4636 E. University Drive, Suite 245
Phoenix AZ 85034
Tel: 602 968-7140
Fax: 602 968-0334

Pioneer-Standard Electronics
4908 E. McDowell Road, Suite 103
Phoenix AZ 85008-4228
Tel: 602 683-4109
Fax: 602 231-8877

California

Agoura Hills

Pioneer-Standard Electronics
5126 Clareton Drive, Suite 160
Agoura Hills CA 91301
Tel: 818 865-5800
Fax: 818 865-5814

Future Electronics
27489 West Agoura Road, Suite 310
Agoura Hills CA 91301
Tel: 818 879-1234
Fax: 818 879-5200

Chatsworth

Anthem Electronics, Inc.
9131 Oakdale Avenue
Chatsworth CA 91311
Tel: 818 775-1333
Fax: 818 775-1302

Irvine

Anthem Electronics, Inc.
1 Oldfield Drive
Irvine CA 92718-2809
Tel: 714 768-4444
Fax: 714 768-6456

Bell Industries
220 Technology Drive, Suite 100
Irvine CA 92618
Tel: 714 727-4500
Fax: 714 453-4610

Future Electronics
25B Technology Way, Suite 200
Irvine CA 92618
Tel: 714 453-1515
Fax: 714 453-1226

Pioneer-Standard Electronics
217 Technology Drive, Suite 110
Irvine CA 92618
Tel: 714 753-5090
Fax: 714 753-5074

Los Angeles

Bell Industries
11812 San Vicente Blvd., Suite 300
Los Angeles CA 90049
Tel: 310 826-2355
Fax: 310 447-3265

Rocklin

Anthem Electronics, Inc.
580 Menlo Drive, Suite 8
Rocklin CA 95765
Tel: 916 624-9744
Fax: 916 624-9750

Roseville

Bell Industries
3001 Douglas Blvd., Suite 205
Roseville CA 95661
Tel: 916 781-8070
Fax: 916 781-2954

Future Electronics
755 Sunrise Avenue, Suite 150
Roseville CA 95661
Tel: 916 783-7877
Fax: 916 783-7988

San Diego

Aegis Electronic Group, Inc.
1015 Chestnut Avenue, Suite G2
Carlsbad CA 92008
Tel: 619 729-2026
Fax: 619 729-9295

Anthem Electronics, Inc.
9369 Carrol Park Drive
San Diego CA 92121
Tel: 619 453-9005
Fax: 619 546-7893

Bell Industries
5520 Ruffin Road, Suite 205
San Diego CA 92123
Tel: 619 576-3290
Fax: 619 492-9826

Future Electronics
5151 Shoreham Place, Suite 220
San Diego CA 92122
Tel: 619 625-2800
Fax: 619 625-2810

Pioneer-Standard Electronics
9449 Balboa Avenue, Suite 114
San Diego CA 92123
Tel: 619 514-7700
Fax: 619 514-7799

© 1997 Microchip Technology Inc.

California (Continued)

San Jose Area

Anthem Electronics, Inc.
1350 McCandless Drive
Milpitas CA 95035
Tel: 408 935-4600
Fax: 408 935-4688

Bell Industries
1161 N. Fairoaks Avenue
Sunnyvale CA 94089
Tel: 408 734-8570
Fax: 408 734-8875

Future Electronics
2220 O'Toole Avenue
San Jose CA 95131
Tel: 408 434-1122
Fax: 408 433-0822

Pioneer-Standard Electronics
333 River Oaks Parkway
San Jose CA 95134
Tel: 408 954-9100
Fax: 408 954-9113

Westlake Village

Bell Industries
125 Auburn Court, Suite 110
Westlake Village CA 91362
Tel: 805 373-5600
Fax: 805 496-7340

Colorado

Anthem Electronics, Inc.
373 Inverness Drive South
Englewood CO 80112
Tel: 303 790-4500
Fax: 303 790-4532

Bell Industries
9351 Grant Street. Suite 460
Thornton CO 80229
Tel: 303 280-1115
Fax: 303 280-0005

Future Electronics
12600 W. Colfax Avenue
Suite B110
Lakewood CO 80215
Tel: 303 232-2008
Fax: 303 232-2009

Pioneer-Standard Electronics
5600 Greenwood Plaza Blvd., Suite 201
Englewood CO 80111
Tel: 303 773-8090
Fax: 303 773-8194

Connecticut

Anthem Electronics, Inc.
61 Mattatuck Heights Road
Waterbury CT 06705
Tel: 203 575-1575
Fax: 203 596-3232

Future Electronics
700 W. Johnson Avenue
Westgate Office Center
Cheshire CT 06410
Tel: 203 250-0083
Fax: 203 250-0081

Pioneer-Standard Electronics
Two Trap Falls, Suite 101
Shelton CT 06484
Tel: 203 929-5600
Fax: 203 929-9791

Westgate Office Center
700 West Johnson Ave.
Cheshire CT 06410
Tel: 203 250-1319
Fax: 203 250-1359

Florida

Altamonte Springs

Anthem Electronics, Inc.
598 S. Northlake Blvd., Suite 1024
Altamonte Springs FL 32701
Tel: 407 831-0007
Fax: 407 831-6990

Bell Industries
650 South Northlake Blvd., Suite 400
Altamonte Springs FL 32701
Tel: 407 339-0078
Fax: 407 339-0139

Future Electronics
237 S. Westmonte Drive, Suite 307
Altamonte Springs FL 32714
Tel: 407 865-7900
Fax: 407 865-7660

Pioneer-Standard Electronics
337 South-North Lake, Suite 1000
Altamonte Springs FL 32701
Tel: 407 834-9090
Fax: 407 834-0865

Deerfield Beach

Future Electronics
1400 E. Newport Center Drive, Suite 200
Deerfield Beach FL 33442
Tel: 954 426-4043
Fax: 954 426-3939

Pioneer-Standard Electronics
674 S. Military Trail
Deerfield Beach FL 33442
Tel: 954 428-8877
Fax: 954 481-2950

Fort Lauderdale

Anthem Electronics, Inc.
5440 NW 33rd Avenue, Suite 101
Ft. Lauderdale FL 33309
Tel: 954 484-0990
Fax: 954 484-0951

Largo

Future Electronics
2200 Tall Pines Drive, Suite 108
Largo FL 33771
Tel: 813 530-1222
Fax: 813 538-9598

© 1997 Microchip Technology Inc.

Distributors

Georgia

Anthem Electronics, Inc.
3305 Breckinridge Blvd., Suite 108
Duluth GA 30136
Tel: 770 931-3900
Fax: 770 931-3902

Bell Industries
3850 Holcomb Bridge Rd., Suite 110
Norcross GA 30092
Tel: 770 446-2333
Fax: 770 446-7264

Future Electronics
3150 Holcomb Bridge Road, Suite 130
Norcross GA 30071
Tel: 770 441-7676
Fax: 770 441-7580

Pioneer-Standard Electronics
4250 C Rivergreen Pkwy
Duluth GA 30136
Tel: 770 623-1003
Fax: 770 623-0665

Illinois

Anthem Electronics, Inc.
1300 Remington Road, Suite A
Schaumburg IL 60173
Tel: 847 884-0200
Fax: 847 884-0480

Bell Industries
175 West Central Road
Schaumburg IL 60195
Tel: (630) 202-6400
Fax: (630) 202-5850

Future Electronics
3150 W. Higgins Road, Suite 115
Hoffman Estates IL 60195
Tel: 847 882-1255
Fax: 847 490-9290

Pioneer-Standard Electronics
2171 Executive Drive, Suite 200
Addison IL 60101
Tel: 630 495-9680
Fax: 630 495-9831

Indiana

Fort Wayne

Bell Industries
525 Airport North Office Park
Fort Wayne IN 46825
Tel: 219 422-4300
Fax: 219 490-2400

Pioneer-Standard Electronics
237 Airport N. Office Park
Fort Wayne IN 46825
Tel: 219 489-0283
Fax: 219 489-6262

Indianapolis

Bell Industries
6982 Hilldale Ct.
Indianapolis IN 46268
Tel: 317 875-8200
Fax: 317 875-8219

Future Electronics
8425 Woodfield Crossing, Suite 175
Indianapolis IN 46240
Tel: 317 469-0477
Fax: 317 469-0448

Pioneer-Standard Electronics
9350 N. Priority Way W. Drive
Indianapolis IN 46240
Tel: 317 573-0880
Fax: 317 573-0979

Kansas

Future Electronics
10977 Granada Lane, Suite 210
Overland Park KS 66211
Tel: 913 498-1531
Fax: 913 498-1786

Maryland

Anthem Electronics, Inc.
7168 Columbia Gateway Drive, Suite A
Columbia MD 21046
Tel: 410 995-6640
Fax: 410 290-9862

Bell Industries
8945 Guilford Road, Suite 130
Columbia MD 21046
Tel: 410 290-5100
Fax: 410 290-8006

Future Electronics
6716 Alexander Bell Drive, Suite 101
Columbia MD 21046
Tel: 410 290-0600
Fax: 410 290-0328

Pioneer-Standard Electronics
15810 Gaither Drive
Gaithersburg MD 20877
Tel: 301 921-3822
Fax: 301 921-3858

Seymour Electronics
Columbia Business Center
6440 Dobbin Road, Suite B
Columbia MD 21045
Tel: 410 992-7474
Fax: 410 992-7410

Massachusetts

Anthem Electronics, Inc.
200 Research Drive
Wilmington MA 01887
Tel: 508 657-5170
Fax: 508 657-6008

Bell Industries
100 Burrt Road, Suite 106
Andover MA 01810
Tel: 508 474-8880
Fax: 508 474-8902

Future Electronics
41 Main Street
Bolton MA 01740
Tel: 508 779-3000
Fax: 508 779-3050

Pioneer-Standard Electronics
44 Hartwell Avenue
Lexington MA 02173
Tel: 617 861-9200
Fax: 617 863-1547

© 1997 Microchip Technology Inc.

Michigan

Grand Rapids
Future Electronics
4505 Broadmoor S.E.
Grand Rapids MI 49512
Tel: 616 698-6800
Fax: 616 698-6821

Pioneer-Standard Electronics
4467 Byron Ctr. Rd. S.W.
Grand Rapids MI 49509
Tel: 616 534-3145
Fax: 616 534-3922

Detroit Area
Future Electronics
35200 Schoolcraft Road, Suite 106
Livonia MI 48150
Tel: 313 261-5270
Fax: 313 261-8175

Pioneer-Standard Electronics
44190 Plymouth Oaks Drive
Plymouth MI 48170
Tel: 313 416-2157
Fax: 313 416-2415

Minnesota

Bloomington
Bell Industries
9401 James Avenue South, Suite 142
Bloomington MN 55431
Tel: 612 888-7747
Fax: 612 888-7757

Eden Prairie
Anthem Electronics, Inc.
7690 Golden Triangle Drive
Eden Prarie MN 55344
Tel: 612 944-5454
Fax: 612 944-3045

Future Electronics
10025 Valley View Road, Suite 196
Eden Prairie MN 55344
Tel: 612 944-2200
Fax: 612 944-2520

Pioneer-Standard Electronics
7625 Golden Triangle
Eden Prairie MN 55344
Tel: 612 944-3355
Fax: 612 944-3794

Thief River Falls
Digi-Key Corporation
P.O. Box 677
701 Brooks Avenue South
Thief River Falls MN 56701
Tel: 218 681-6674
Fax: 218 681-3380

Missouri

Future Electronics
12125 Woodcrest Executive Drive
Suite 220
St. Louis MO 63141
Tel: 314 469-6805
Fax: 314 469-7226

Pioneer-Standard Electronics
4227 Earth City Express Way
Earth City MO 63045
Tel: 800 344-5218
Fax: 314 209-3054

New Jersey

Northern
Anthem Electronics, Inc.
26 Chapin Road
Pine Brook NJ 07058
Tel: 201 227-7960
Fax: 201 227-9246

Bell Industries
271 Route 46 West
Suites F202-203
Fairfield NJ 07004
Tel: 201 227-6060
Fax: 201 227-2626

Future Electronics
1259 Route 46 East
Parsippany NJ 07054
Tel: 201 299-0400
Fax: 201 299-1377

Phase 1 Technology Corp.
295 Molnar Drive
Elmwood Park NJ 07407
Tel: 201 791-2990
Fax: 201 791-2552

Pioneer-Standard Electronics
14A Madison Road
Fairfield NJ 07006
Tel: 201 575-3510
Fax: 201 575-3454

Seymour Electronics
520 Fellowship Road, Suite 104
Mt. Laurel NJ 07854
Tel: 201 465-0890
Fax: 201 465-0890

Southern
Bell Industries
158 Gaither Drive, Suite 110
Mt. Laurel NJ 08054
Tel: 609 439-8860
Fax: 609 439-9009

Future Electronics
12 East Stow Road, Suite 200
Marlton NJ 08053
Tel: 609 596-4080
Fax: 609 596-4266

New Mexico

New York

Binghamton
Pioneer-Standard Electronics
1249 Front Street, Suite 201
Binghamton NY 13901
Tel: 607 722-9300
Fax: 607 722-9562

Long Island
Anthem Electronics, Inc.
47 Mall Drive
Commack NY 11725
Tel: 516 864-6600
Fax: 516 493-2244

Future Electronics
801 Motor Parkway
Hauppauge NY 11788
Tel: 516 234-4000
Fax: 516 234-6183

Phase 1 Technology Corp.
46 Jefryne Blvd.
Deer Park NY 11729
Tel: 516 254-2600
Fax: 516 254-2693

Pioneer-Standard Electronics
Meridian Plaza III
3125 Veterans Memorial Highway
Ronkonkoma NY 11779
Tel: 516 738-1700
Fax: 516 738-1790

Seymour Electronics
357 Crossways Park Drive
Woodbury NY 11797
Tel: 516 496-7474
Fax: 516 496-0857

Upstate New York
Future Electronics
300 Linden Oaks
Rochester NY 14625
Tel: 716 387-9550
Fax: 716 387-9563

Future Electronics
200 Salina Meadows Pkwy., Suite 130
Syracuse NY 13212
Tel: 315 451-2371
Fax: 315 451-7258

Pioneer-Standard Electronics
1250 Pittsford/Victor Road
Building 200
Pittsford NY 14534
Tel: 716 389-8200
Fax: 716 389-8240

© 1997 Microchip Technology Inc.

Distributors

North Carolina

Anthem Electronics, Inc.
4805 Green Road, Suite 100
Raleigh NC 27604
Tel: 919 871-6200
Fax: 919 790-8970

Bell Industries
3100 Smoketree Court, Suite 800
Raleigh NC 27604
Tel: (919) 874-0011
Fax: (919) 874-0013

Future Electronics
8401 University Executive Park
Suite 108
Charlotte NC 28262
Tel: 704 547-1107
Fax: 704 547-9650

Future Electronics
5225 Capital Blvd.
1 North Commerce Center
Raleigh NC 27604
Tel: 919 790-8782
Fax: 919 790-8782

Pioneer-Standard Electronics
2200 Gateway Centre Blvd., Suite 215
Morrisville NC 27560
Tel: 919 460-1530
Fax: 919 460-1540

Ohio

Cleveland

Bell Industries
31200 Solon Road, Suite 12
Solon OH 44139
Tel: 216 498-2002
Fax: 216 498-2006

Future Electronics
6009-E Landerhaven Drive
Mayfield Heights OH 44124
Tel: 216 449-6996
Fax: 216 449-8987

Pioneer-Standard Electronics
4800 East 131st Street
Cleveland OH 44105
Tel: 216 498-6305
Fax: 216 587-3906

Pioneer-Standard Electronics
29125 Solon Road
Solon OH 44139
Tel: 216 248-8710
Fax: 216 248-9166

Dayton

Bell Industries
444 Windsor Park Drive
Dayton OH 45459
Tel: 513 435-5922
Fax: 513 435-3122

Future Electronics
1430 Oak Court, Suite 203
Beavercreek OH 45430
Tel: 513 426-0090
Fax: 513 426-8490

Pioneer-Standard Electronics
8741 Gander Creek Drive
Miamisburg OH 45342
Tel: 513 428-6900
Fax: 513 428-6995

Worthington

Pioneer-Standard Electronics
100 Old Wilson Bridge, Suite 105
Worthington OH 43085
Tel: 614 848-4854
Fax: 614 848-4889

Oklahoma

Pioneer-Standard Electronics
9717 E. 42nd Street, Suite 105
Tulsa OK 74146
Tel: 918 665-7480
Fax: 918 665-1891

Oregon

Anthem Electronics, Inc.
11515 S.W. Sequoia, Suite 160
Portland OR 97224
Tel: 503 598-9660
Fax: 503 598-7893

Bell Industries
8705 S.W. Nimbus, Suite 100
Beaverton OR 97008
Tel: 503 644-3444
Fax: 503 520-1948

Future Electronics
7204 S.W. Durham Road, Suite 900
Portland OR 97224
Tel: 503 603-0956
Fax: 503 603-0955

Pioneer-Standard Electronics
5665 S.W. Meadows Road, Suite 150
Lake Oswego OR 97035
Tel: 503 968-6565
Fax: 503 598-2555

Pennsylvania

Philadelphia Area

Anthem Electronics, Inc.
355 Business Center Drive
Horsham PA 19044
Tel: 215 443-5150
Fax: 215 675-9875

Pioneer-Standard Electronics
500 Enterprise Road
Keith Valley Business Center
Horsham PA 19044
Tel: 215 674-4000
Fax: 215 674-3107

Pittsburgh

Pioneer-Standard Electronics
259 Kappa Drive
Pittsburgh PA 15238
Tel: 412 782-2300
Fax: 412 963-8255

© 1997 Microchip Technology Inc.

Texas

Austin

Anthem Electronics, Inc.
14050 Summit Drive, Suite 119
Austin TX 78728
Tel: 512 388-0049
Fax: 512 388-0271

Bell Industries
1044 Camino La Costa, Suite 3122
Austin TX 78752
Tel: (512) 452-1127
Fax: (512) 452-2818

Future Electronics
6850 Austin Center Blvd., Suite 610
Austin TX 78731
Tel: 512 502-0991
Fax: 512 502-0740

Pioneer-Standard Electronics
1826-D Kramer Lane, Suite D
Austin TX 78758
Tel: 512 835-4000
Fax: 512 835-9829

Dallas

Anthem Electronics, Inc.
651 North Plano Road, Suite 401
Richardson TX 75081
Tel: (972) 238-7100
Fax: (972) 238-0237

Bell Industries
14110 Dallas Parkway, Suite 170
Dallas TX 75240
Tel: (972) 458-0047
Fax: (972) 404-0267

Future Electronics
800 E. Campbell Road, Suite 130
Richardson TX 75081
Tel: (972) 437-2437
Fax: (972) 669-2347

Pioneer-Standard Electronics
13765 Beta Road
Dallas TX 75244
Tel: (972) 386-7300
Fax: (972) 490-6419

Houston

Future Electronics
10333 Richmond Avenue, Suite 970
Houston TX 77042
Tel: 713 785-1155
Fax: 713 785-4558

Pioneer-Standard Electronics
10530 Rockley Road, Suite 100
Houston TX 77099
Tel: 713 495-4700
Fax: 713 495-5642

San Antonio

Pioneer-Standard Electronics
8200 I-10 West, Suite 705
San Antonio TX 78230
Tel: (210) 377-3440
Fax: (210) 377-3626

Utah

Anthem Electronics, Inc.
1279 West 2200 South
Salt Lake City UT 84119
Tel: 801 973-8555
Fax: 801 973-8909

Bell Industries
6912 S. 185 West, Suite B
Midvale UT 84047
Tel: 801 561-9691
Fax: 801 255-2477

Future Electronics
3540 S. Highland Drive, Suite 301
Salt Lake City UT 84106
Tel: 801 467-4448
Fax: 801 467-3604

Washington

Anthem Electronics, Inc.
19017 - 120th Avenue, N.E., Suite 102
Bothell WA 98011
Tel: 206 483-1700
Fax: 206 486-0571

Bell Industries
19119 Borth Creek Pkwy, Suite 102
Bothell WA 98011
Tel: 206 646-8750
Fax: 206 646-8559

Future Electronics
19102 N. Creek Parkway
Suite 118
Bothell WA 98011
Tel: 206 489-3400
Fax: 206 489-3411

Pioneer-Standard Electronics
2800 156th Avenue S.E.
Suite 100
Bellevue WA 98007
Tel: 206 644-7500
Fax: 206 644-7300

Wisconsin

Bell Industries
W. 226 N. 900 Eastmound
Waukesha WI 53186
Tel: 414 547-8879
Fax: 414 547-6547

Pioneer-Standard Electronics
120 Bishop's Way, Suite 163
Brookfield WI 53005
Tel: 414 784-3480
Fax: 414 784-8207

Future Electronics
250 North Patrick Blvd., Suite 170
Brookfield WI 53045
Tel: 414 879-0244
Fax: 414 879-0250

© 1997 Microchip Technology Inc.

Distributors

© 1997 Microchip Technology Inc.

MICROCHIP

Factory Sales

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602-786-7200 Fax: 602-786-7277
Technical Support: 602 786-7627
Web: http://www.microchip.com

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 972-991-7177 Fax: 972-991-8588

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 714-263-1888 Fax: 714-263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 416
Hauppauge, NY 11788
Tel: 516-273-5305 Fax: 516-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong
Microchip Asia Pacific
RM 3801B, Tower Two
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

India
Microchip Technology India
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-0061 Fax: 91-80-229-0062

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai
Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hongiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700
Fax: 86 21-6275-5060

Singapore
Microchip Technology Taiwan
Singapore Branch
200 Middle Road
#10-03 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886 2-717-7175 Fax: 886-2-545-0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44-1628-851077 Fax: 44-1628-850259

France
Arizona Microchip Technology SARL
Zone Industrielle de la Bonde
2 Rue du Buisson aux Fraises
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 München, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleone
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-39-6899939 Fax: 39-39-6899883

JAPAN

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shin Yokohama
Kohoku-Ku, Yokohama
Kanagawa 222 Japan
Tel: 81-4-5471- 6166 Fax: 81-4-5471-6122

5/8/97

12

© 1997 Microchip Technology Inc.



MICROCHIP

Printed in the USA, 5/97
© 1997, Microchip Technology Inc.

Microchip Technology Inc. • 2355 W. Chandler Blvd. • Chandler, Arizona USA • 602-786-7200 • FAX: 602-899-9210

© 1997 Microchip Technology Inc.