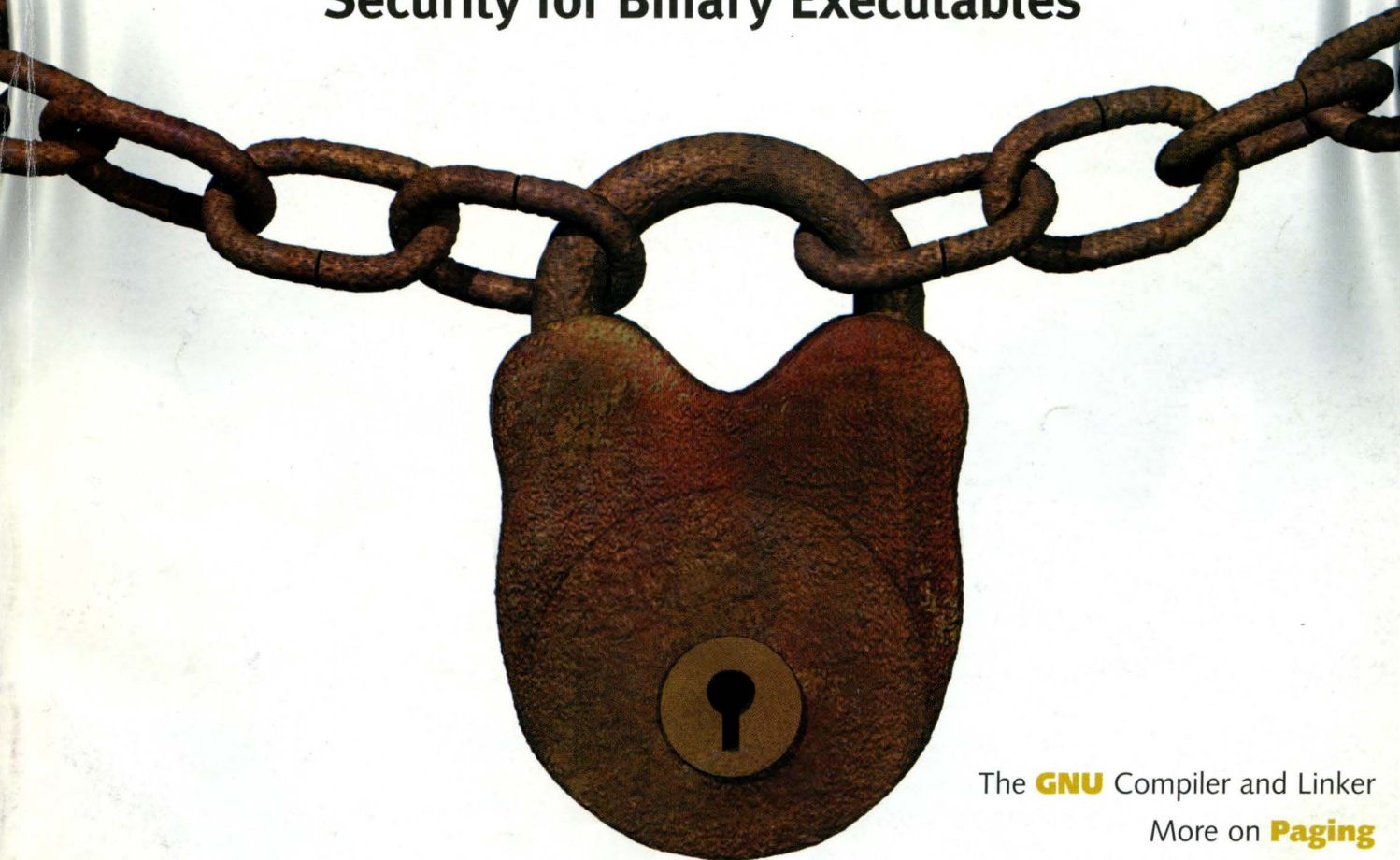


Embedded Systems[®]

P R O G R A M M I N G

Protecting Firmware

Security for Binary Executables



The **GNU** Compiler and Linker

More on **Paging**

A **Paeon** to Noise

Crenshaw's Divine **Hammer**



Internet Appliance Design: Speech Coding Algorithms

Cut Development Time

Using In-Circuit Emulators and BDM's



NEW
seeHau
User Interface

Macros

- Macro debug capability
- Project support
- Command line
- Shadow RAM support
- Single step in your macro
- GUI capable macros

Key Features

- Real-time emulation at maximum chip speeds
- High level support for popular C compilers
- Advanced tracing capabilities
- Configurable user interface with remote hookup capability
- High speed connection to PC through parallel port (LPTx) or plug-in ISA card

Nohau Supports These Microcontroller Families

**8051 80C196 683xx 68HC11 P51XA ST10
MCS296 MCS251 68HC16 C166 68HC12 M16C**

www.nohau.com
noHau

51 East Campbell Avenue, Campbell, CA 95008
Phone: 1-888-88NOHAU (1-888-886-6428)
Fax: 408-378-7869 E-mail: sales@nohau.com

Final Assault

Seek and destroy even the most resilient hardware and software bugs with EST's visionICE emulators, visionPROBE JTAG/BDM cables and visionCLICK C/C++ debuggers.

EST's arsenal of tools supports all PowerPC,[™] ColdFire[®] and CPU32 processors. It's tightly integrated with VxWorks[®], pSOS[™], and many other RTOSs.

Fast. Painless. Lethal.



visionICE



visionCLICK



visionPROBE

PowerPC[™] **68K[®] COLD FIRE[®]**

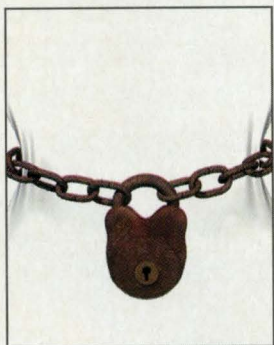
EST

Embedded Support
Tools Corporation

Call 800-957-5588 or Download a Free Demo: www.estc.com

contents

F E B R U A R Y 2 0 0 0



COVER

Protecting your binary executables from reverse engineering by competitors isn't a matter to be taken lightly these days.

Cover illustration by Rupert Adley.

Cover Story

24

Protecting Binary Executables

Any company that develops embedded products must make a substantial investment of resources to get to the product stage. This article surveys the techniques available for protecting the resulting binary executable from hackers and/or reverse engineering by potential or current competitors.

BY MATT FISHER

66

Embedding with GNU: The GNU Compiler and Linker

The GNU compiler, gcc, is capable of producing high-quality code for embedded systems of all types. Here the author discusses the gcc features that are most useful for embedded engineers.

BY BILL GATLIFF

Final Part

80

Data Memory Paging Management, Part 2

This article, which began last month, explains a method to detect any potential paging errors in assembly programs.

BY HUGH O'BYRNE



80

Detect and find potential paging errors using one programmer's detailed method.

Featured Section

internet appliance design

33 CONNECTING... Virtual Serial Ports

What if you need a serial port when there are none, or all of them are already being used for other purposes? Here are some tips on creating virtual serial ports from other hardware that you may already have at your disposal.

BY MICHAEL BARR

39 Implementing Speech Coding Algorithms

This article describes some techniques you can use to implement speech coding standards in a real-time system.

BY DAVID H. CRAWFORD AND
EMMANUEL ROY

55 Embedding with XML

The eXtensible Markup Language (XML) is more flexible than HTML, which makes it appropriate in embedded systems that use web technologies.

BY SCOTT LAWRENCE

61 Embedded Internet Tools

New internet appliance design products.

columns

7 PROGRAMMER'S TOOLBOX On Hammers and Nails

BY JACK W. CRENSHAW

63 PROGRAMMING POINTERS Top-Level cv-Qualifiers in Function Parameters

BY DAN SAKS

95 SPECTRA A Paean to Noise

BY DON MORGAN

113 BREAK POINTS Reality Bites

BY JACK G. GANSSE

117 STATE OF THE ART Finding C Level

BY P.J. PLAUGER



103

The Rabbit2000
eight-bit micro-
processor from
Rabbit
Semiconductor

departments

5 #INCLUDE Beginnings and Endings

Whether this year marks the end of an era or the beginning of a new one might be debatable, but changes abound regardless.

BY LINDSEY VEREEN

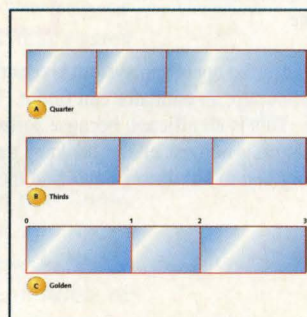
103 NEW PRODUCT GALLERY

Rhapsody in MicroC from I-Logix;
SurroundView from Accelerated Technology;
Slim740 ChipConnect from Mitsubishi;
DProbeTriCore Level 2 emulator from Hitex
Development Tools; more

106 CAREERS

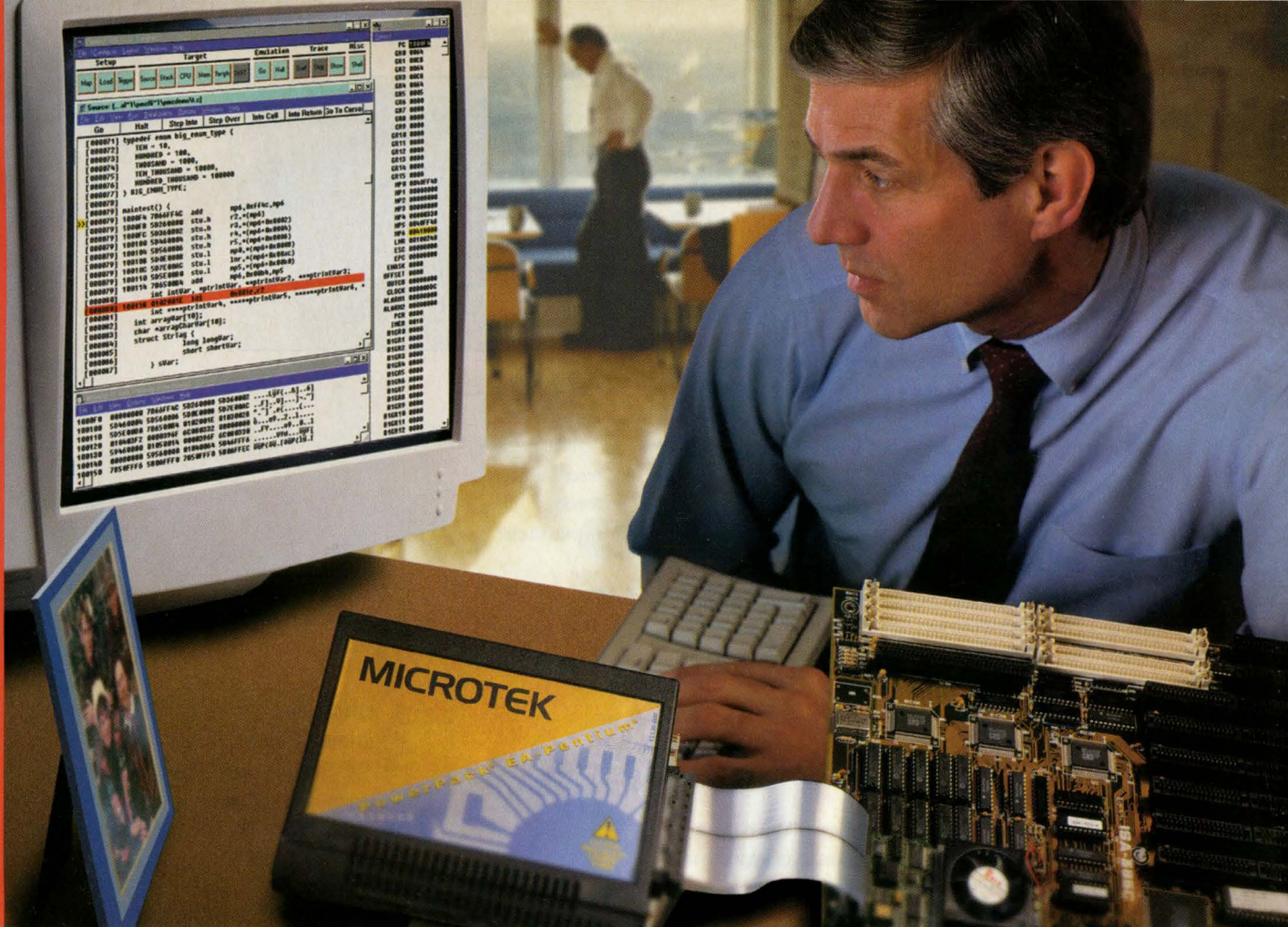
108 MARKETPLACE

112 ADVERTISER INDEX



7

Explore the
mysteries of the
Golden Ratio,
courtesy of Jack
Crenshaw.



"If only I had spent less time on debug..."

I remember when I budgeted for my project. I refused to spend money on debug tools. I didn't think I would need them. The money I saved would make me a hero.

Now, my project is one month, two months ... six months overdue. Nobody thinks I'm a hero. What would I give to get some of that time back?

Microtek debug tools can make time. They speed up development and testing, saving you precious time at the end of the project, where you need it most!

Microtek emulators offer the debug features you need to track down and correct software, hardware, and system integration issues. They can find the errors software debuggers cannot see.

First, emulators can debug before the operating system is functioning. If an unexpected issue is affecting boot up, an emulator can find it. They are also operational after a hard crash. This is significant, because software debuggers lose debug information. Microtek emulators keep track of the last 128 KB of bus cycles, allowing you to sift through and find the problem.

URGENT!

If you are developing a Pentium class target using a reference design or off-the-shelf board, please take a moment to speak with our staff. They will insure your design is on track, and tool friendly if debug is needed in the future.

Finally, if there is a subtle issue during the integration of hardware, Microtek EA emulators are capable of providing a clock-cycle-by-clock-cycle trace that allows you to view each signal and determine whether it was in the correct state. Microtek's tools are clearly superior to software debug solutions.

There has never been a project that couldn't use more time when it's critical ... like in final testing when everything has come together ... or in final debug, when everything is going down in flames.

Wouldn't it be great to deliver your next project on time? Next time, be sure to put a Microtek emulator in your project plan from the start.

It will help take the knot out of your stomach. And it will improve your company's bottom line!

MICROTEK
IN-CIRCUIT EMULATORS

1 (800) 886-7333

Phone (503) 533-4463

Fax (503) 533-0956

Email - info@microtekintl.com

Additional interfaces: CAD/UL® and Windriver Tornado II®

Financing Available

www.microtekintl.com



Lindsey Vereen

EDITORIAL DIRECTOR
Lindsey Vereen, lvereen@mfi.com

MANAGING EDITOR
Michael Shapiro, mshapiro@mfi.com

TECHNICAL EDITOR
Michael Barr, mbarr@mfi.com

SPECIAL PROJECTS EDITOR
Tarita Whittingham, twhittingham@mfi.com

ASSISTANT EDITOR
Felisa Yang, fyang@mfi.com

CONSULTING TECHNICAL EDITORS
Jack G. Gansle
Jerome L. Krasner, PhD

CONTRIBUTING EDITORS
Jack W. Crenshaw
Larry Mittag
Don Morgan
P.J. Plauger
Dan Saks

PRESIDENT/ELECTRONICS
Steve Weitzner

VICE PRESIDENT/ELECTRONICS
Donna Esposito

EMBEDDED/DSP GROUP DIRECTOR
Mike Flynn, (415) 278-5251

PUBLISHER
Eric Berg, (415) 278-5220

CALIFORNIA SALES MANAGER
Andres Diaz, (415) 278-5274

CALIFORNIA SALES ASSISTANT
Molly Bruns, (415) 278-5298

WESTERN ACCOUNT EXECUTIVE
Craig Hammond, (415) 278-5294

SALES ASSOCIATE, RECRUITMENT
Sam Louis, (415) 278-5223

EASTERN REGIONAL SALES MANAGER
Damon Graff, (781) 235-8258

EASTERN SALES
Jared Grimm, (781) 235-8258

PRODUCTION COORDINATOR
James Whitehead

CIRCULATION MANAGER
Jennifer Schuler

CIRCULATION DIRECTOR
John W. Rockwell

SUBSCRIPTION CUSTOMER SERVICE
Toll Free: (888) 847-6177
Non-U.S. subscribers: (847) 647-8602
embedded@halldata.com
Back issues may be purchased on a pre-paid basis through: Miller Freeman, 1601 West 23rd St., Suite 200, Lawrence, KS 66046; (800) 444-4881; (785) 841-1631

REPRINTS
Sherry Bloom, (415) 808-3980

EXECUTIVE PRODUCER, EMBEDDED.COM
Leticia Smith, lsmith@mfi.com

ONLINE PRODUCTION COORDINATOR
Billy Biondi, wbiondi@mfi.com

PRESIDENT/CEO, CMP MEDIA INC.
Gary Marshall

EXECUTIVE VICE PRESIDENTS
Regina Starr Ridley John Russell
Steve Weitzner Tony Uphoff

**SENIOR VICE PRESIDENT/
GLOBAL SALES & MARKETING**
Bill Howard

**SENIOR VICE PRESIDENT/
BUSINESS DEVELOPMENT**
Pam Watkins



Visit our Web site at
www.embedded.com

Beginnings and Endings

In the December issue, I remarked that my editorial would be my last of the millennium. A veritable plethora of readers instantly turned on me. One grumbled:

"By now I am resigned to the fact that the general media refer to 2000 as the start of the next century and millennium. However, it is really very sad to see this mistake made in a technical magazine. Can't you do better than this? On the bright side maybe your last article of the millennium will be able to avoid having such a gaff."

Ouch. And that was one of the gentler letters I received. Ten years ago (that is, January 1990) I acknowledged the discrepancy between what was popularly perceived as the beginning of the new decade and what a close analysis of the calendar and a reflection on history would seem to indicate. I observed that for those who acknowledge the least significant bit (LSB) to be zero, then the least significant year (LSY) ought to be zero as well; hence, no problem.

Okay, that argument might have been slightly specious. Nevertheless, this time around, I just yielded to popular thinking and assumed that the millennium began last month. Actually, it would be better if the millennium does begin next year, because the parties will be much less expensive than they were this time around. You couldn't find any venue that didn't soak you for two or three times the normal New Year's rates for the privilege of celebrating the alleged millennium.

Actual millennium or not, a few changes are in store for *Embedded Systems Programming* this year. As you may know, several months ago Miller Freeman's parent company, United News & Media plc, acquired CMP, the publisher of *EETimes*. Beginning this month, *Embedded Systems Programming* will be

published under the CMP brand and will be part of the group that includes *EETimes*, *Integrated System Design*, *Communication Systems Design*, and *Electronic Buyers' News*. This is a swell match for us. The *EETimes* folks et al. are an unbeatable bunch, talented, smart, knowledgeable about the industry, and great writers to boot, and we look forward to significant cross pollination. This union will result in some major changes to our Internet presence as well, as will become evident over the next several months.

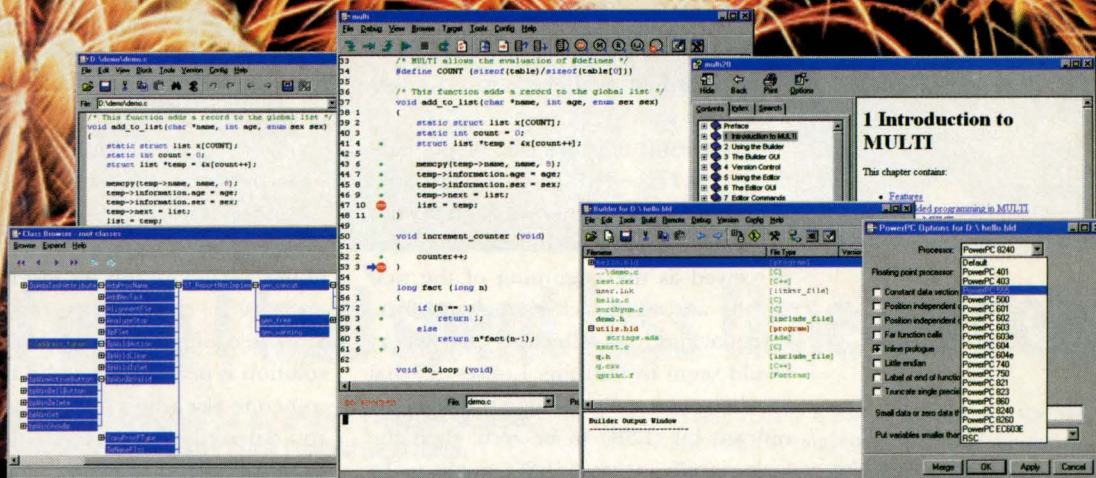
Another change I expect to see in the near future is the addition of a West Coast technical editor for *Embedded Systems Programming*. Michael Barr is doing a bang-up job on the East Coast, and we'd clone him for the San Francisco office if we could. Since that solution is problematical, I'd like to find someone else who's just as good. I guess you'd describe the perfect candidate as a bit of a teacher, a bit of a writer, and a bit of a project manager, with the heart and soul of an engineer. So if you have an inclination to make the leap into the fast-paced, multitasking, and altogether wacky world of publishing, now is your chance. It would be the perfect way either to start the new millennium or to end the old one, depending on your proclivities.

Lindsey Vereen

lvereen@mfi.com

MULTI[®] 2000

The New IDE for the New Millennium



- New and Improved GUI
- Graphical Browser
- On-Line Help
- Syntax Coloring and Auto Indenting

- Simulator
- Version Control
- C++ Debugging
- EventAnalyzer

- Project Builder
- Code Coverage
- Run-Time Error Checking
- Profiling

The Best Compilers.

The Best Debuggers.

Now, all wrapped up in a brand new, powerful, easy to use IDE. Once you try MULTI 2000, you won't want to struggle with "old fashioned" tools ever again. Contact us now for a Free Evaluation CD and

start the millennium with a bang!



Tel: 805.965.6044 • Fax: 805.965.6343 • Email: sales@ghs.com • www.ghs.com

Copyright ©1999 Green Hills Software, Inc. MULTI is a registered trademark of Green Hills Software, Inc.



Jack W. Crenshaw

On Hammers and Nails

"To a person whose only tool is a hammer, every problem looks like a nail," goes an old saying. To belabor the obvious, the saying has a negative connotation. The idea is that if you only know how to do things a certain way, you're going to try to use that method for every problem you encounter. Sometimes it will be appropriate. Most times, it won't.

The following story is 100% gospel truth: my father's favorite tool was the hacksaw. I'm not talking about a mild infatuation here; he was head-over-heels in *love* with hacksaws, and saw every problem as a new opportunity to put one to work. He once sawed off a stub of water pipe, flush with the concrete floor. When I suggested maybe this wasn't one of his better ideas, he assured me that the pipe was part of the old pump system, and no longer had water pressure in it.

He was wrong.

Our house had those old-fashioned (meaning, in this case, superior to today's) doorknobs that were adjustable for end play. The doorknob on the door to my mother's room had gotten loose, and she asked my father to fix it. He took one look at the doorknob, wiggled it a few times, then turned to me and said, "Son, go fetch my hacksaw."

My mother and I screamed in horror, and I ended up adjusting the knob myself, as well as every subsequent mechanical problem around that house.

Maybe that was his plan all along.

Thinking about it, I realize that the hammer/nail saying would make a pretty good motto for this column.

The column's title is, after all, "Programmer's Toolbox," and my purpose from the get-go has been to provide you readers with as many and as varied a collection of tools as possible. The general idea is that the more tools you have, the more likely you are to have the right one for the job.

In a way, this column is all about craftsmanship. The average homeowner's toolbox has a hammer. The

- Recognizes which tool best fits the problem
- Knows when a new tool is needed

For the record, this concept also applies well to one's choice of programming language. We've all met people who are outspoken advocates of one language or another. You've got your C fanatics, your C++ fanatics, your Visual BASIC fanatics, and, per-

Optimization should not be like trying to drive a square peg into a round hole with the wrong kind of hammer.

handyman's toolbox may have a claw hammer and a ball-peen hammer. The craftsman's toolbox has those, plus a tack hammer, a rubber mallet, a wooden mallet, several styles of auto body hammer, three sizes of sledgehammer, and so on. You get the idea. Though it takes skill to be a craftsman, it also takes a well-equipped toolbox.

You have to have both the skills and the toolbox. Give my father a handyman's toolbox, and he would use only the hacksaw. On the other hand, Stradivarius could never have made those exquisite violins if his only tool was a bent screwdriver. You have to have both.

A craftsman:

- Has a good collection of tools
- Knows how to use them all
- Knows when and where to use them

haps the most fanatic of all, aficionados of Fortran, APL, and other obscure languages. Each claims benefits for his or her favorite language that can't be achieved in any other.

Way back in the dark ages of personal computers, one of the most egregious *misuses* of a tool came from programmers in Microsoft BASIC. The Microsoft BASIC for the Radio Shack TRS-80 had a feature whereby you could enter assembly-language subroutines, byte by byte, as *decimal* integers, then execute them. Some TRS-80 programmers prided themselves in writing BASIC programs using hundreds, if not thousands, of bytes of executable "data." Today, some people spend all their time, and even make professional careers out of, writing macros for Microsoft Word and Excel.

Of course, you, dear readers, would

Of all the reasons I can think of for using a given language for a given problem, the worst possible reason has to be: "It's the only one I know."

never fall into this trap. Your toolbox has more than one language in it because, as software craftsmen, you know that each tool has its uses, you know the advantages and disadvantages of each, and you know when and how to use each one. Of all the reasons I can think of for using a given language for a given problem, the worst possible reason has to be: "It's the only one I know."

Lately, however, it's occurred to me that the story about the hammer and the nails has, like the nail itself, another end to it. Lately, it seems that I've been encountering, in both my work and my play, a nail of a particular new shape. And here I stand, with the wrong kind of hammer. I'm talking about optimization.

Lately, I've encountered quite a few problems that seem to involve fitting functions to observed data, for one reason or another. Naturally, we'd like to make the fit as "good" as possible, which usually implies minimizing some kind of error criterion.

Normally, such problems would be a natural for the least-squares fit, also known as "linear regression," or sometimes even "nonlinear regression." However, none of the problems I've been bumping into lately lend themselves very well to least-squares solutions. Although I'm sure there must be other solutions I haven't thought of yet, at least I've been able to see that a good, general-purpose function optimizer can and will pound these new-fangled nails into submission. I have a gut feeling that once we have a good set of function optimizers in our respective toolboxes, we'll be seeing a whole new class of nails, where we once saw only thorns. We may even find that they get used as often as my father's proverbial hacksaw.

Hence our current mission to study the techniques for function minimization.

The story so far

In the last few months, we've been looking at ways to find the minimum of a function, $f(x)$, of a single variable. We got as far as a method based on searching the function space by halving intervals, when we ran into a question of robustness. That question led me into a digression, last month, to seek a method that behaved itself when the function in question was being coy, and trying to hide from us the true location of the minimum.

Specifically, we worried about situations not often considered in other texts, where two or more samplings of the function yield exactly the same value for $f(x)$. Though we might feel justified in claiming that exactly equal values are extremely unlikely using floating-point arithmetic, I showed you some examples last month, hardly pathological cases, where *exactly* equal values were not only likely, but almost inevitable.

We spent most of last month addressing this problem, and seeking a way to search the problem space without having an elusive minimum slip from our clutches. I found a solution, which is based on "wiggling" one of the sampling points in an attempt to force the minimum out of hiding. Having done so, we're now ready to proceed with the mainstream process of developing a good, general-purpose tool. Or perhaps a set of them.

Where were we?

When we left off last month, we had arrived at a pretty robust method for searching for a minimum of a function:

$$y = f(x) \quad (1)$$

given three starting points which "bracket" the function. By that I mean that we start with three points P_0 , P_2 , and P_4 such that:

$$y_2 < y_0 \text{ and } y_2 < y_4 \quad (2)$$

Starting from this configuration guarantees us that there is at least one minimum in the interval $x_0..x_4$, and the algorithm is designed to home in on it by reducing the size of that interval without letting the minimum slip away. The method involved halving each interval, testing the value of $f(x)$ at the new points P_1 and P_3 , and choosing three of the five points such that the relation in Equation 2 is maintained. In that way, we inexorably tighten the net around the minimum until it's found within some acceptable epsilon range of x .

(By the way, I apologize for the confusion in naming these points. In last month's article and previously, I've been calling them $P_1..P_5$. The code, however, shows them as $P_0..P_4$. We tried to make the change in editing last month, to keep the nomenclature consistent, but we missed the edit deadline. I'll try to be consistent, and use the latter nomenclature from now on.)

I thought you'd like to see the performance of our halving algorithm, for our standard test function:

$$f(x) = \cos(2\pi x^3) \quad (3)$$

The exact value of the minimum is given by:

$$x_{\min} = \sqrt[3]{\frac{1}{2}} = 0.7937005259841 \quad (4)$$

Table 1 shows the five points used at each iteration. The highlighted points, x_1 and x_3 , are the ones generated anew during that iteration. I've chosen to show the values as fractions rather than decimal numbers, so you can best see the bisection process in operation. To see it even better, convert each fraction to the same denominator, and you'll see that the numerators always vary by one, across the row. After nine iterations, we have the minimum bracketed between 203/256

ONE GIANT LEAP FOR MANKIND. (AND MILLIONS OF LITTLE STEPS FOR THE ENGINEERS)



30 years ago the American Space Program landed a man on the moon for the first time, and millions watched it happen live. The Infineon Technologies Space Program isn't quite so glamorous – but we do have our moments. Today, our development tools partners are working together to make the design-in process for our 8- and 16-Bit Microcontrollers, TriCore and DSP products easier for you to work with and use in your next application. For more information on our tools, turn to the next 3 pages, or visit us on our web site: www.spacetools.com



PHILIPS



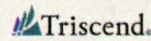
AMD

ARM

NEC



ARC



Hands-on Computer Labs in Embedded Systems Applications, Chips & Tools!

SIMPLIFY THE DESIGN PROCESS AND GET YOUR BREAKTHROUGH PRODUCT TO MARKET QUICKER!

This is your chance to evaluate dozens of the leading embedded microcontrollers and their development tool chains—in an actual college laboratory environment!



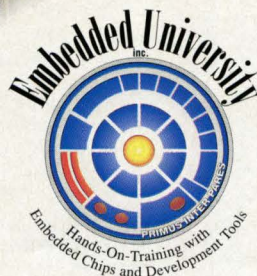
April 3-7, 2000

Mission College
Technology Center
Santa Clara, CA.

A FULL WEEK OF 2-HOUR TUTORIALS ON THE HOTTEST NEW EMBEDDED APPLICATIONS:

Embedded Internet, Industrial & Motor Control, USB, Communications, Routers, Information and Appliances, and CAN etc.

PLUS: • WindowsCE Workshops • Panel Discussions on EEMBC, NEXUS, System on a Chip, etc. • Discussions on Trends in Embedded Systems Development by Jerry Krasner • Analysis by Tom Starnes and Jim Turley on the newest microcontroller trends • and more!



Applications

Take Labs & Tutorials on THE hot applications, including:
Embedded Internet
System Integration
Universal Serial Bus
Wireless
Motor Control
...and dozens more!

Special Topics

Select from daily seminars and panel discussions on:
Programming of Flash Micros
OPAMPs Embedded in 8-Bit Micros
Benchmarking 8-Bit Compilers
Debugging Motorola MCU Designs
Writing Software for ARM Designs
Create your own CSOC
Selection of Low-Cost MCUs
And more to choose from!

Chips

AMD 186, Elan, and K6
Infineon Technologies C500,
C166, TriCore
Philips 8051, XA
STMicroelectronics ST6, ST7, ST9,
ST10, STPC
ARM All Families
Zilog Z8
Mitsubishi M16C, Slim 740
NEC V800, 78K
Triscend E5 Configurable SOC
ARC Cores Ltd. 32-Bit Processor
Analog Devices Microconverter
National Semiconductor COP8
Microchip PIC 16
Hitachi H8
Motorola 68HC08, M-Core
.....AND MORE!

Tools

Development Tools from the leading suppliers worldwide presented together in test chains!
• Compilers
• Assemblers
• Debuggers
• RTOS
• Emulators
• IDEs
• Logic Analyzers
...and Dozens of tools designed for special applications
Work with the experts in computer labs designed specifically for your needs!

PORTABLE
Design

Embedded Systems

Register On-line Today and Receive FREE Evaluation Boards, a Visor Delux or PalmV

www.EmbeddedU.com

TRACE32®

IN-CIRCUIT-EMULATORS & DEBUGGERS

ONE SYSTEM FITS ALL!



- Bond-Out Emulators and OCDS Level 1 & 2 Debuggers
- Support for TriCore, UTAH, EGOLD, C161..C167 and C500 Family
- Download your free Simulator debugger For TriCore

Embedded
Systems 2000
Nürnberg
Feb. 16-18, 2000

ESC Spring 2000
Chicago
Feb. 28-Mar. 02, 2000

Lauterbach Datentechnik GmbH
Fichtenstraße 27
D-85649 Hofolding, Germany
Fax: ++49-8104-8943-49
Phone: ++49-8104-8943-0
e-mail: info@lauterbach.com

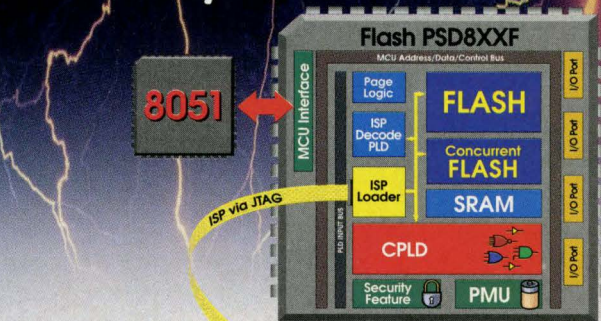
Lauterbach, Inc.
4, Mount Royal Ave
Marlborough, MA 01752
Fax: 508-303-6813 • Phone: 508-303-6812
email: info_us@lauterbach.com

LAUTERBACH

<http://www.lauterbach.com>

8051... need Flash?

We have your Flash solution



Program Flash for the first time, everytime, or anytime right on the board!



Add up to 256KB of Flash, 32KB of EEPROM or Flash Boot, up to 64Kbit of SRAM, programmable logic, additional I/O, power management, and advanced In-System Programming (ISP) capabilities to your next 8051 based embedded control system with a single chip called an **EasyFLASH®** PSD. They are the first memory device to offer In-System Programming the first time for programming at the end of the assembly line, and In-Application re-Programming for fast "on-the-fly" field upgrades. Visit our website www.waferscale.com to request further information, and to download your free copy of our new "Point and Click" configuration software ... **PSDsoft Express™**.

www.waferscale.com/8051.html

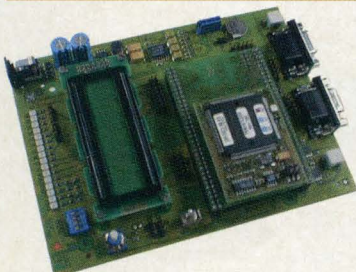


Waferscale-USA
Tel. 800-832-6974
Fax 510-657-5916
info@waferscale.com

Waferscale-EUROPE
Tel. 33-1-69320120
Fax 33-1-69320219
wsifrance@compuserve.com

Infineon Technologies 8-, 16-, and 32-Bit Hardware & Software Development Tools Support

Evaluation + Application Board Starterkit *STK16x.500*



"Plug & Play" Starterkit
160 x 120 mm EVA-Board
Cable included
Power Supply included
2 * 16 Character LCD-Display
16 LED for I/O Line Indication
DIP-Switch for Digital Input
A/D Conversion
Stepper Motor Circuit
2 CAN Driver
free Demo Programs with open C-Code
powerful Download Tools including TQMinimodul

all in one:

Digital Input
Digital Output
Analog Input
PWM
LCD
Stepper Motor Control
Serial Interface (SIO)
CAN
10BaseT Ethernet

supported by:

Ing.-Büro Dr. Kanef
Systemsoftware, Echtzeitsystemen

KEIL SOFTWARE
pls
Programmierbare Logik & Systeme

TASKING

SAB-C 164CI
SAB-C165
SAB80C166
SAB-C167LM
SAB-C167CR-LM
SAB-C167CS-32FM

TQComponents

Fax: (49) 8153-9308-34 • eMail: info@tqc.de • Internet: www.tqc.de

DProbe167 In-circuit Emulator



Small ...

What is small enough to hold in one hand but has enough functionality to get the job done ... fast! Our DProbe167! Through sophisticated ASIC technology the DProbe167 can go with the developer anywhere easily and functionality is not compromised. The DProbe167 is also upgradeable with its modular design. And even the difficulties of adaptation are made simple with our revolutionary PressOn technology.

These are not the only reasons why we have the most C166 emulators installes around the world. Contact us today and we will tell you more about how the DProbe167 can help you get ahead in today's competitive world.

... but smart!

Hitex Development Tools
710 Lakeway Drive, Suite 280
Sunnyvale, CA 94086

Tel.: (800) 45HITEX
Tel.: (408) 733 7080
Fax: (408) 733 6320
E-mail: info@hitex.com
Internet: www.hitex.com

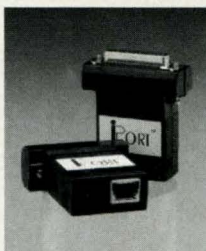
hitex
DEVELOPMENT TOOLS

I²C Bus and SMBus Tools

MCC

**MICRO
COMPUTER
CONTROL**

PO Box 275
Hopewell, NJ 08525
USA
Tel: (609) 466-1751
Fax: (609) 466-4116
Email: info@mcc-us.com

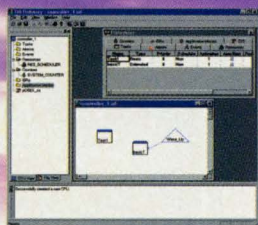


- I²C/SMBus Monitors
- Protocol Analyzers
- Host Adapters
- Smart Battery Tools
- Keypad-LCD-I/O ICs
- Prototyping Boards
- Dist. & Mux. Boards

VISIT OUR WEB SITE
TO LEARN MORE ...

www.mcc-us.com

Let pOSEK™ lead you into the future



- Super-small scalable kernel
- OSEK/VDX 2.0 compliant
- Minimal RAM/ROM requirements
- 16 & 32 bit processor Support
- Industry-leading design & development tools



Detailed information at: www.pOSEK.com

© 1999 Integrated Systems, Inc., pOSEK is a registered trademark of Integrated Systems, Inc.

KEIL
SOFTWARE

**Development
Tools**

µVISION 2

C51

C251

C166

**IDE, Compiler, Debugger,
Real-time OS & more...**

Keil Software develops, manufactures, and distributes software development tools for the 8051, 251, and 166 microcontroller families. Development tools include C compilers, assemblers, real-time kernels, debuggers & simulators, integrated environments, and evaluation boards. Our web site provides up-to-date information about our development tools, evaluation software, updates, application notes, and example programs.

In America:

Keil Software, Inc.
16990 Dallas Parkway, Suite 120
Dallas, TX 75248-1903
☎ 800-348-8051 / 972-735-8052

In Europe and Asia:

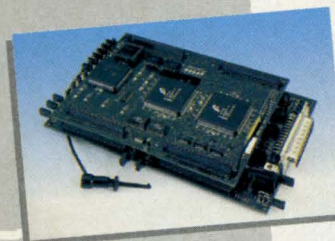
Keil Elektronik GmbH
Bretonischer Ring 15
D-85630 Grasbrunn - Munich
☎ ++49 89 456040-0

www.keil.com

Infinion Technologies 8-, 16-, and 32-Bit Hardware & Software Development Tools Support

EMUL166™-PC

**In-Circuit
Emulator
for
Siemens
C166 &
C500
Family**



Nohau's EMUL166-PC supports the Infineon Technologies C166 family of microcontrollers, including C167CR, C167SR, C165, C163, C161RI and their associated Xperipherals. The EMUL166-PC emulator system consists of a card which connects to the PC's printer port (LPTx:), and a pod (containing the controller, memory and logic) and a cable connecting the two. The pod consists of two PCB boards with the ability to connect to a third board for tracing capabilities. The optional trace board can record and trigger on internal ROM and external buses. External trigger in/out is also available.

EMUL51™-PC

supports the Infineon Technologies C500 family including the C505C, C505CA and C515C.

NOHAU
CORPORATION

51 E. Campbell Ave.
Campbell, CA 95008-2053
TEL: (408) 866-1820
FAX: (408) 378-7869
E-Mail: sales@nohau.com
URL: <http://www.nohau.com>

EMUL166-PC Features:

- Emulates at maximum chip speed (33MHz)
- High-level support for all popular C-compilers
- Set hardware and software breakpoints
- 256K to 1M external pod memory available
- Customizable register windows
- Optional trace board

MULTI
2000

The New IDE for the New Millennium



The Best Compilers • The Best Debuggers

Now, all wrapped up in a brand new, powerful, easy to use IDE. Once you try MULTI 2000, you won't want to struggle with "old fashioned" tools ever again. Contact us now for a Free Evaluation CD and start the millennium with a bang!

Green Hills
SOFTWARE INC.

US Headquarters: 805.965.6044
European Headquarters: +44 (0)1494 429336
Email: sales@ghs.com • www.ghs.com

Beat your Competition to the Tape...



with PHYTEC Rapid Development Kits

All you need for **Rapid Development**:

- Insert-ready PHYTEC Single Board Computer module:
- microMODUL (36 x 51 mm.) supports the, **C501, C504, C505C, C165** and **C166**
- miniMODUL (55 x 85 mm.) supports the **C509, C515C, C517, 80C535, 80C537, C166** and **C167CR**
- nanoMODUL (38 x 47 mm.) supports the **C164CI**
- to 256 KB SRAM, to 256 KB Flash, serial interfaces
- Development Board with AC adapter and DB-9 serial interfaces
- Controller User's Manual, QuickStart Manual and schematics
- PHYTEC Spectrum CD-ROM including software development tools (C Compiler, Assembler, Debugger and Monitor), example programs, extensive documentation and AppNotes
- PHYTEC FlashTools download utility

PHYTEC

Stick It In!

www.phytec.com

PHYTEC America LLC
255 Ericksen Avenue NE
Bainbridge Island, WA 98110
Tel: (800) 278-9913
Fax: (206) 780-9135

PHYTEC Messtechnik GmbH
Robert-Koch-Str. 39
55129 Mainz GERMANY
Tel: +49 (06131) 9221-0
Fax: +49 (06131) 9221-22



IF IT WEREN'T FOR THE CMX RTOS, WE'D HAVE TO FIND A DIFFERENT USE FOR MICROCONTROLLERS.

Most people agree that microcontrollers without development tools are just worthless chunks of silicon—until you add the CMX-RTX Real-Time Multi-Tasking Operating System!

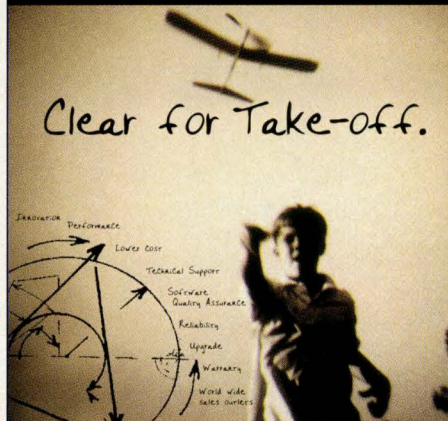
CMX-RTX Supports Most 8-, 16-, 32-Bit Processors & DSPs

For Infineon we support:
8051, 80C16x & TriCore

- TCP/IP stack also Available •



680 Worcester Road,
Framingham MA 01702
PH: (508) 872-7675
email: cmx@cmx.com
FAX: (508) 620-6828
WWW: www.cmx.com



EMBEDDED DEVELOPMENT TOOLS FOR:

- **Phillips:** 80C51, 80C51XA, Smart Cards, Contactless Cards
- **Infineon:** TriCore, C500
- **Mitsubishi:** M16C, 37700
- **Motorola:** 68HC05, 68HC08, 68HC11
- **OKI:** OLMS-66K

EUROPEAN DISTRIBUTOR FOR
YOC ADVISE PRODUCTS FOR:
ARM, Hitachi, NEC, Fujitsu,
Mitsubishi, Motorola,
Toshiba

Member of IEEE-ISTO-5001
(NEXUS) Consortium



<http://www.ashling.com>

Infineon Technologies 8-, 16-, and 32-Bit Hardware & Software Development Tools Support

TCP/IP FOR C16x

Fast, compact embedded TCP/IP
networking solutions for C16x

- TCP/IP protocols
- Embedded Web Server
- Internet Access Package
- SNMP
- Supports PHYTEC KitCON and Keil Toolchain

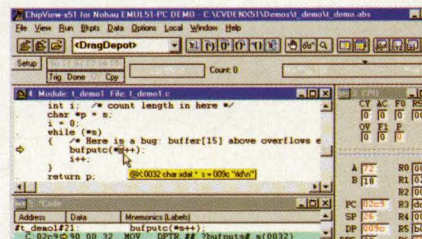


U.S. SOFTWARE
EMBEDDED EXCELLENCE

7175 NW Evergreen Pkwy. Suite 100 Hillsboro, OR 97124
TEL: 503-844-6614 • FAX: 503-844-6480
EMAIL: info@ussw.com • www.ussw.com

CHIPVIEW-x51

THE DEBUGGER OF CHOICE BY 8051 PROGRAMMERS!



- Full Simulator, ROM monitor and ICE versions
- Custom support for NOHAU's EMUL51-PC
- Turbo Debug Power plus Windows ease-of-use



CHIPTOOLS
Software Tools for Hardware Solutions

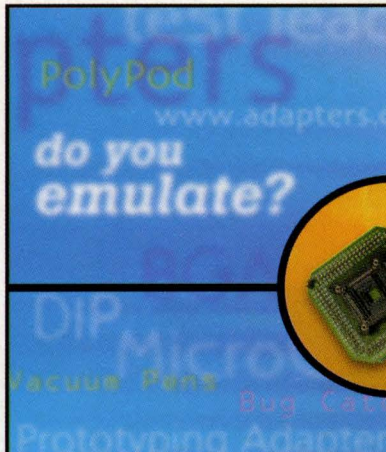
Tel: (905)274-6244
Fax: (905)891-2715
info@chiptools.com

Visit us at: www.chiptools.com
for a Free Demo today!

HI-TECH C Compiler 8051

- ✓ Fast code
- ✓ Full ANSI C
- ✓ Reliable code
- ✓ Remote debugger
- ✓ Mature - 14 years!
- ✓ Excellent support
- ✓ Economical - \$850!
- ✓ Money back guarantee

Call Now!
800 735 5715
Fax 407 722 2902
www.htsoft.com
sales@htsoft.com



Then chances are you need an adapter for your emulator. We provide Interconnect solutions for any device, any IC package, or any application.

- Fast and accurate tech-support.
- The industry's most experienced design team.
- Need quick turn? How does two weeks sound!



Emulation Solutions
Phone: 408.855.8527
www.adapters.com

and $813/1024$, which is, in decimal, the range:

0.79296875 .. 0.794921875

The width of this interval is $1/512$, or 0.00195. Perhaps more to the point, the middle value is only 0.0002 from the true solution. So far, we have narrowed the width of the range containing the solution to give us only the first two digits, but we're definitely homing in on the solution. Only 11 more iterations should reduce the error to one part in a million, which is about all one can hope for from a function minimizer. You can see, then, that the term "slow" is a relative one. Much faster methods may be out there, but you have to admit, 20 iterations is not an unreasonably large number, especially for such a simple algorithm.

Four-point bisection

The bisection algorithm isn't perfect, but it certainly does work, especially when I use the tweak I call "wiggling" the new point to resolve ties in the function values. Crude as it is, the possibility of the minimum escaping our grasp is virtually nil. The biggest problem with the method is that we perform two new function evaluations per iteration. That's certainly better than the 10 we had not too many issues ago, but it's exactly twice as many as we need.

Let's not forget that we can't push any minimization algorithm too far in our quest for a solution. The nature of the problem assures that we can't expect to get six, eight, or even 15 digits of accuracy in the solution. At the limit, as we approach the minimum, every continuous function looks like a parabola, and the closer we get to the solution, the flatter and flatter that parabola gets, until it's indistinguishable from a straight line. At that point, we can't say where the solution lies; we can only say that it must be between x_0 and x_4 . This is not the

As long as we aren't too greedy, we can inexorably close the noose around the minimum until we can be sure we have it corralled tightly enough. The minimum cannot escape our grasp.

most satisfying solution in the world; as computer types, we're used to getting answers back that are accurate to as many digits as the floating-point format allows.

However, in this case, you definitely must get used to disappointment. The nature of the problem simply won't allow you the satisfaction of such accuracy. If our algorithm is robust, as I think this one is, we can guarantee that the minimum we seek will not escape our grasp

is between x_0 and x_4 , and my best guess as to its location is halfway between them." Thus, to this algorithm, it's the reduction in the interval—the tightening of the noose—that is the important thing. Looking at Table 1, it's easy to see how this noose is tightened at every iteration. Because the process is predictable, we can easily estimate how many iterations we'll need to get a satisfactory solution.

Once we recognize that the whole

TABLE 1 Points used in bisection

Iteration	x_0	x_1	x_2	x_3	x_4
0	0	1/4	1/2	3/4	1
1	1/2	5/8	3/4	7/8	1
2	3/4	13/16	7/8	15/16	1
3	3/4	25/32	13/16	27/32	7/8
4	3/4	49/64	25/32	51/64	13/16
5	25/32	101/128	51/64	103/128	13/16
6	101/128	203/256	51/65	205/256	103/128
7	101/128	405/512	203/256	407/512	51/64
8	405/512	811/1024	203/256	813/1024	407/512
9	203/256		813/1024		407/512

because of incorrect decisions made in choosing the bracketing points. As long as we aren't too greedy, we can inexorably close the noose around the minimum until we can be sure we have it corralled tightly enough. The minimum cannot escape our grasp. However, we *can* most definitely *squeeze* it out, if we try to tighten the noose too much.

Later in this series, we'll look at methods that use quadratic interpolation to give our best guess as to the actual location of the minimum. The bisection method, however, doesn't use such an approach. With this method, we are not so much interested in the location of the minimum itself, as the size of the range $x_0..x_4$ that brackets it. At the end of the iteration, we can only say, "The minimum

purpose of the exercise is to bring x_0 and x_4 closer together, we should ask ourselves: how many function evaluations will it take to do this at every step? Clearly, the answer is that we can get by with only one. As an added bonus, we are required to maintain only four internal data points, rather than five.

The trick is to recognize that we don't need to compute both x_1 and x_3 in a single iteration. We can do them on alternate iterations. Note carefully that we *must* bisect both the left-hand and right-hand intervals. The only way we're going to reduce the bracketing interval, $x_0 .. x_4$, is to move both end points inward. But we don't have to do both motions in a single step. We can do them on alternate steps.

The code of Listing 1 does this.

LISTING 1 Single bisection

```

/* Perform one step of a minimization by single
 * bisection. Input required is three points, of
 * which the middle one is strictly smaller.
 * The points need not be evenly spaced.
 */
void bisect(double (*f)(double), double &x0, double &y0,
           double &x2, double &y2,
           double &x4, double &y4,
           int count)
{
    double x_mid, y_mid;

    // alternate bisection on even and odd values
    if(count & 1)
    {
        // Bisect the first interval
        x_mid = (x0+x2)/2;
        y_mid = f(x_mid);

        // if new point is lower,
        // use as new middle
        if(y_mid < y2)
        {
            x4 = x2;
            y4 = y2;
            x2 = x_mid;
            y2 = y_mid;
            return;
        }

        // if it's higher, move in left limit
        if(y_mid > y2)
        {
            x0 = x_mid;
            y0 = y_mid;
        }
    }
    else
    {
        // Bisect the second interval
        x_mid = (x2+x4)/2;
        y_mid = f(x_mid);

        // if new point is lower,
        // use as new middle
        if(y_mid < y2)
        {
            x0 = x2;
            y0 = y2;
        }
    }
}

```

Here, I've used an iteration counter to tell me whether I should bisect the left-hand or the right-hand interval. For simplicity, I've left out my little "wiggle" tweak, but never fear: we'll put it back for the final version.

In Table 2, I've shown how the function converges, in the same format as Table 1. Comparing the two tables, can you see the main difference? We've reduced the rate of convergence. There's no such thing as a free lunch, and by reducing the number of function evaluations, we've also assured that we get less improvement, in terms of reducing the interval, per iteration.

In Table 1, after nine iterations our denominator was 512. In Table 2 it's 64—a factor of eight smaller. Nevertheless, the algorithm of Listing 1 represents an improvement in performance. Before, we needed 18 function evaluations to get a reduction in interval by 2^{-9} . On average, we reduce the interval by a factor of $\sqrt{2}$, or 1.414, *per function evaluation*. The new algorithm needs nine evaluations to get a reduction by 2^{-6} , which means that, on average, we reduce the interval by a factor of $2^{2/3}$, or 1.588, per function evaluation—definitely an improvement. As we increase the number of iterations, the new algorithm is bound to win, hands down.

What about intervals?

Although the code of Listing 1 works just fine, it leaves us in a peculiar situation with respect to symmetry. When we were using five points, we used bisection to give us four equal-width intervals. Choosing three of the five points for the next iteration gave us two equal-width intervals again, so the solution maintained a certain symmetry. This is not so when we use only four intervals. The situation is as shown in Figure 1a. After the bisection, we end up with an oddly asymmetric configuration. On the average, we can expect the minimum to lie to the left of the centerline half

The Start of Something Big.

Introducing BlueCat™

from Lynx.

Linux Solutions for

Embedded

Systems.



www.bluecat.com

Lynx
REAL-TIME SYSTEMS, INC.



from Lynx
BlueCat
Linux Solutions For Embedded Systems

Of course, one may always argue that beauty is in the eye of the beholder. Some people might like rooms made like bowling alleys, or square rooms.

LISTING 1, continued Single bisection

```

    x2 = x_mid;
    y2 = y_mid;
    return;
}

// if it's higher, move in right limit
if(y_mid > y2)
{
    x4 = x_mid;
    y4 = y_mid;
}
}
}

/* Solve for minimum of function, using n successive bisections.
 * Initial points MUST represent legal configuration, i.e.,
 * y0 > y1, y2 > y1.
 * The points need not be evenly spaced.
 */
double minimize(double (*f)(double), double &x0, double &y0,
               double &x1, double &y1,
               double &x2, double &y2, double eps)
{
    // protection from infinite loop
    int i = 60;
    while((fabs(x2-x0)>eps)&&(i>0))
    {
        bisect(f, x0, y0, x1, y1, x2, y2, i);
        i--;
    }
    if(i==0)
        cout << "iteration failed" << endl;
    return x1;
}

```

TABLE 2 Points used, single bisection

Iteration	x0	x1	x2	x3	x4
0	0		1/2	3/4	1
1	1/2	5/8	3/4		1
2	5/8		3/4	7/8	1
3	5/8	11/16	3/4		7/8
4	11/16		3/4	13/16	7/8
5	3/4	25/32	13/16		7/8
6	3/4		25/32	51/64	13/16
7	25/32	101/128	51/64		13/16
8	101/128		51/64	103/128	13/16
9	101/128		51/64		103/128

the time, so we'll select the two equal-width intervals for the next iteration. The other half, though, leaves us with one interval twice as large as the other.

Although this situation is hardly fatal—on the next iteration, the wider interval will be halved—it does leave us wondering if we've made the right choice for the divisions.

Since we now are using four points instead of five, the obvious solution is to divide the full interval into thirds, as shown in Figure 1b. But that solution doesn't work, either. After the division, we'll end up selecting only two of the three intervals, which are now equal in width. We're back to a bisected situation, and therefore back where we were in Figure 1a.

So where does this lead us? Figure 1a shows us that we can't always have equal-width intervals. Figure 1b says that even if we try to make them so, we'll soon be back to the same situation.

Given that we can't have the equal-width intervals that we'd like, can we at least have a division that gives us the same relationship between intervals, at each iteration? Yes, we can, and that's what Figure 1c is all about.

What makes this geometry so special? Well, if you look at the proportions of the two left-hand intervals, you'll see that it is the same as the larger proportion. Stated another way, the ratio of the distances 0..1 and 1..2 is the same as the distances 0..2 and 2..3. The same is true at the right-hand end. In other words, regardless of which pair of intervals we must choose for the next iteration, the intervals will always have the same ratio. What's more, this ratio will continue to hold for all future divisions.

Golden oldies

You have just met the famous Golden Ratio, sometimes called the Golden Section. It's a world-famous number

whose roots extend back into antiquity. It was certainly known to the ancient Greeks; it appears in much of their architecture. Some folks say it was known to the pyramid builders, as well. It can be found in other Egyptian architecture, too.

The question regarding the Golden Section was originally posed in architecture this way: given that we're going to build a rectangular room, what proportions would be most pleasing to the eye? Some examples are shown in Figure 2. Clearly a long, skinny rectangle like the first example is not going to work. Aside from the obvious problem that there would be no room for the pool table, it just doesn't look right. Likewise, a square is no good—it's too boring. And a rectangle that's too close to a square would have the same problem. The one in the middle is, as Goldilocks said, "just right."

Of course, one may always argue that beauty is in the eye of the beholder. Some people might *like* rooms made like bowling alleys, or square rooms. But it's been asserted down through the ages that most people prefer a room shaped like the middle rectangle. That room has the property that the ratio of the two sides, a/b , is the same as the larger side to the total. In other words,

$$\frac{a}{b} = \frac{b}{a+b} \quad (5)$$

Now, this one equation won't give us enough information to solve for both a and b ; that would be like allowing only one sized room. We can, however, solve for the ratio, by letting $x = b/a$. Some manipulation gives:

$$\frac{1}{x} = \frac{x}{1+x}$$

$$\text{or } x^2 - x - 1 = 0 \quad (6)$$

This is a quadratic equation, and has roots:

FIGURE 1 Intervals; a) quarters, b) thirds, c) golden



A Quarter

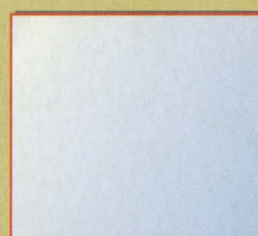
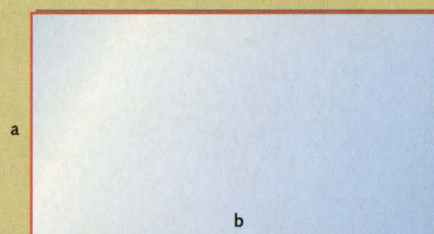
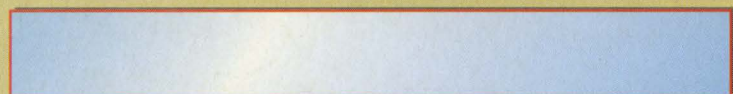


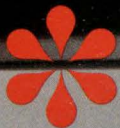
B Thirds



C Golden

FIGURE 2 Rectangles






FUJITSU

HITACHI



華為技術
HUAWEI

 **Italtel**

NEC

OKI

 **3Com®**

Some of the biggest names in the business
come to us for smart networking solutions.
We're glad to chip in.



If you get excited just thinking about the possibilities for networking and computing, you're our kind of designer. And we're happy to connect you with all kinds of smart solutions. DigitalDNA technology from Motorola is more than chips. It's software, systems and the ideas of thousands of innovative engineers dedicated to helping create the next generation of routers, switches, servers, modems, desktops and more. You'll find DigitalDNA in leading-edge technology like 0.10µm copper interconnect CMOS-based solutions. The world's fastest copper interconnect SRAMS. And a wide portfolio of highly scalable PowerPC cores. So whenever you need a smart networking solution, you know the name to turn to. www.digitaldna.motorola.com

POWERPC™ • FSRAM • COMMUNICATIONS PROTOCOL MODULES • DSP • LDMOS/RF • COLDFIRE®

M O T O R O L A E M B E D D E D S O L U T I O N S

LISTING 2 Solution by Golden Section

```

/* Perform one step of a minimization using the golden ratio.
 * Input required is three points, of
 * which the middle one is strictly smaller.
 * The points need not be evenly spaced.
 */
void golden(double (*f)(double), double &x0, double &y0,
            double &x2, double &y2,
            double &x4, double &y4){
{
    double x_mid, y_mid;
    static const double golden = (sqrt(5)-1)/2;

    // Divide the first interval
    x_mid = x0 + golden*(x2-x0);
    y_mid = f(x_mid);

    // if new point is lower,
    // use as new middle
    if(y_mid < y2)
    {
        x4 = x2;
        y4 = y2;
        x2 = x_mid;
        y2 = y_mid;
        return;
    }

    // if it's higher, move in left limit
    if(y_mid > y2)
    {
        x0 = x_mid;
        y0 = y_mid;
    }
}

/* Solve for minimum of function, using n successive divisions
 * based upon the golden section.
 * Initial points MUST represent legal configuration, i.e.,
 * y0 > y1, y2 > y1.
 * The points need not be evenly spaced.
 */
double minimize(double (*f)(double), double &x0, double &y0,
                double &x1, double &y1,
                double &x2, double &y2, double eps)
{
    double temp;

    // protection from infinite loop
    int i = 60;

```

$$\frac{1-\sqrt{5}}{2} \quad \text{and} \quad \frac{1+\sqrt{5}}{2} \quad (7)$$

Since the first root gives a negative number, we'll take the second, and write:

$$x = \frac{1+\sqrt{5}}{2} = 1.618033989 \quad (8)$$

This number is called the Golden Ratio, and is usually denoted by ϕ . It has some fascinating properties. For example:

$$\begin{aligned} \phi^2 &= \phi + 1 \\ \frac{1}{\phi} &= \phi - 1 \end{aligned} \quad (9)$$

The number pops up in all kinds of interesting places. Pick any number at random. Add one to it, and take the reciprocal. Add one to that, and take the reciprocal again. Keep doing that long enough, and you arrive at ϕ . Or, take the ratio of any two successive terms in the Fibonacci series:

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

This ratio also converges to ϕ . The ratio appears in the most surprising places, such as the number of petals on some flowers, or the sizes of the chambers in a seashell. A philosopher might argue that the reason things proportioned according to ϕ is pleasing to our eyes is because the number is fundamental to nature.

So what does all this have to do with finding function minima? Simply that the proportions of Figure 1c also are obtained from the Golden Section. Given the full interval from 0 to 3, the interval from 0 to 2 is given by the ratio $\phi-1$. Likewise, the interval from 0 to 1 is gotten by scaling 0..2 by the same ratio.

Imagine we go into the minimum-seeking function formerly known as `bisect()` with the three points 0, 2, and 3. We get the new point, 1, by the formula:



Some things are instinctive. (Now including logic analyzers.)

Only \$3,200*

LogicWave

- Debug designs with 8- and 16-bit microcontrollers
- 100 MHz state analysis and 250 MHz timing analysis
- 128 K memory per channel
- 34 channels of logic analysis

Doing what comes naturally. That's the idea behind LogicWave, Agilent Technologies' new logic analyzer. It may be the easiest logic analyzer you've ever used. How? For starters, it's PC-based, for familiar, simple operation. So you won't need to waste hours learning—and relearning—the interface. You can draw the trigger easily by selecting a trigger condition from a drop-down menu. And its low cost means you can actually afford one of your own. Because a logic analyzer isn't very helpful if you have to share it with ten other engineers.



Experience LogicWave for yourself. Visit our web site for a FREE download of the actual interface software. Or call us to get a FREE demo on CD-ROM. And talk to another engineer who'll answer your questions about LogicWave's ease of use. Although, after you've used it once, you probably won't have any.

www.agilent.com/find/bi

1-800-452-4844, Ext. 6918



Agilent Technologies
Innovating the HP Way

The Real Embedded Developer's Toolkit



Solutions for
microprocessors including:

**8051, C166/ST10,
M16C, 196, DSP56xxx,
68xxx, PowerPC,
TriCore & CARMEL**

TASKING Understands the value of
your time. You've got to get
that product to market yesterday,
ahead of schedule - reducing
development costs. Time is money.

Call us today at
1-800-458-8276

visit our web site at:
www.tasking.com

TASKING
Tools to Build your Success

PROGRAMMER'S TOOLBOX ●

However, there's a much cleaner way to accomplish the same thing, and that's to swap points P0 and P3 before each computation. The function thinks it's always dividing the left-hand interval, but in fact it's not.

LISTING 2, continued Solution by Golden Section.

```
while((fabs(x2-x0)>eps)&&(i>0))
{
    golden(f, x0, y0, x1, y1, x2, y2);
    temp = x0;
    x0 = x2;
    x2 = temp;
    temp = y0;
    y0 = y2;
    y2 = temp;
    i--;
}
if(i==0)
    cout << "iteration failed" << endl;
return x1;
}
```

$$x_1 = x_0 + (\phi - 1)(x_2 - x_0) \quad (10)$$

We can make our program do this with a simple one-line change to the function. Or, rather, a two-line change, since the math appears twice, depending on the value of *count*. However, we can do better than that. The use of the *count* parameter has always seemed to me a kludge, anyhow. We had to introduce it to make sure that the function alternated left and right bisections. However, there's a much cleaner way to accomplish the same thing, and that's to swap points P₀ and P₃ before each computation. The function *thinks* it's always dividing the left-hand interval, but in fact it's not.

It may seem like a waste of clock cycles to keep swapping the data around like this. If this bothers you, feel free to go with the longer form as in Listing 1. However, I suspect that the code to execute the test on *count*, and to pass its value into the function,

probably costs more than the data swapping. Anyhow, the new code just looks cool, doesn't it?

The modified code is shown in Listing 2. Next month, I'll clean it up a bit more, put back the "wiggle" gimmick, and add a few other niceties.

A correction

In the December issue, I gave the incorrect Internet URL for Scilab. The correct URL is www-rocq.inria.fr/scilab/scilab.html. Sorry about the mistake. Many thanks to Keith Wright and John B. Williston for straightening me out. **esp**

Jack W. Crenshaw a senior principal design engineer at Alliant Tech Systems Inc. in Clearwater, FL. He did much early work in the space program and has developed numerous analysis and real-time programs. He holds a PhD in physics from Auburn University. Crenshaw enjoys contact and can be reached via e-mail at jcrens@earthlink.net.

INTEL CPU PLATFORM

WIDE AREA NETWORKING

SYSTEM PLATFORMS

DSP TECHNOLOGIES

**HERE'S A NO-NONSENSE MESSAGE ABOUT
OUR TOTAL TELECOM SOLUTION.
DON'T BE FOOLED BY ITS SIMPLICITY.**



Only RadiSys offers the total telecommunications solution.

A solution this complete is hard to pull off. A lot of vendors offer different parts of telecom solutions. Only RadiSys has the vision and technical expertise to deliver the total, integrated solution across both PCI and CompactPCI architectures. RadiSys has acquired Texas Micro, and is now positioned to redefine how the market is buying OEM embedded telecommunications systems. Now you can let us put the total solution together. Download our comprehensive white paper on SS7, and find out how simple it is to get the total solution from one source at www.radisys.com, or call 1-877-837-6859.

INTEL CPU PLATFORM

State of the art Pentium solutions for Pentium II and Pentium III architectures

SYSTEM PLATFORMS

Enterprise and carrier class CompactPCI and PCI system enclosures and backplanes

WIDE AREA NETWORKING

Field-proven frame relay, SS7, ATM, T1/E1, X.25, IP and other protocols and interfaces

DSP TECHNOLOGIES

Advanced TI C6x voice processing solutions-hardware and algorithms

Download FREE white paper today!
www.radisys.com/SS7

RadiSys



Protecting Binary Executables

Any company that develops embedded products must make a substantial investment of resources to get to the product stage. This article surveys the techniques available for protecting the resulting binary executable from hackers and/or reverse engineering by potential or current competitors.

Companies in the embedded industry usually make substantial investments of time, money, and intellectual resources to solve typical design problems. The result of a successful project is a product in which the company probably has a longer-term financial interest. Protecting embedded software from unwanted reverse engineering by potential competitors or hackers is generally of the utmost importance.

The purpose of this article is to survey some options for protecting binary executables stored in a product's EPROM. This is an introductory, high-level treatment of the subject; additional techniques surely exist that I won't cover in this survey. With this article I intend to foster awareness of the subject and to pique your interest to explore further if your circumstances warrant. The survey of protection techniques follows a brief overview of decompilation.

Following the survey I'll present two scenarios for which some of these techniques may prove worthwhile.

Decompilation

Most *ESP* readers are familiar with the compilation process. Compilation enables programmers to write software in a high-level language, expressing program flow in a way that is closer to how the human mind works—more abstract than what today's microprocessors can handle. The high-level language/compiler combination reduces the abstraction to a set of specific instructions the microprocessor can understand.

Decompilation, then, is simply the reverse process: taking the low-level instructions executed by the processor and backing out the higher-level intent of the human programmer. More specifically, decompiling is the process of generating source-level code from the executable object code

Decompiling is the process of generating source-level code from the executable object code embedded in a product's memory. It is a matter of reverse engineering the software.

embedded in a product's memory. It is a matter of reverse engineering the software.

Certainly the high-level intent of an application can be at least partially deduced by simply using the product. The user interface, system inputs and outputs, and general product behavior can be readily observed by any end user. Additionally, the architectures and instruction sets for commercial microprocessors are publicly available. Therefore, we can assume that someone who is familiar with your product and the microprocessor it uses already knows a good deal about your application, whether or not they have access to the software. This person is familiar with the processor's addressing modes, on-chip peripherals, address space, and so forth. So even though real-time interactions and algorithms are "hidden" in the object code, they can be recovered to some degree (maybe to a large degree) by decompiling the executable object code, which is often stored in a read-only memory (ROM) chip. This information could then be used to your competitive disadvantage.

What follows is a survey of possible techniques to discourage the would-be reverse engineer. Almost all of these techniques will only slow down the decompiling effort; they will not keep the reverse engineer from succeeding, given enough time, resources, and desire.

Methods of protection

Use a copyright notice in the binary file. This is certainly not a technical protection scheme—only a legal one. But including an ASCII copyright notice in the binary image at least notifies anyone reading the image that the software is legally protected and should not be tampered with. A copy-

right notice will not hinder an ethically challenged competitor or hacker, but it only takes up a handful of bytes in ROM, so little reason exists not to include it.

Use a checksum value over the whole application. This method allows you to detect modifications to the executable image. You'll need a boot program, which stores the correct checksum for your binary image. The boot program itself should reside in nonvolatile, protected memory, preferably inside of the microprocessor itself if it has on-board ROM of some type.

The boot program must also be able to calculate a checksum from the ROM image, and compare the result to the stored value. If the two values differ, the boot program inhibits execution of the application. This method offers no protection against reading and decompiling the code; it only protects against putting an altered ROM in the product and having the product still function.

Note that neither the boot program nor the stored checksum should be part of the ROM image that you're trying to protect. If the boot program can be located and decompiled, then this method offers no protection at all. That's why locating the boot program within the microprocessor itself is a good idea—the program is much more difficult to retrieve that way. Many of today's microcontrollers offer some on-board ROM or flash which can be used for this purpose. The checksum can be stored in EEPROM or flash.

Encrypt string data. If your product has any sort of human user interface, the software likely includes text strings. Compilers typically store the string data in ASCII format, and by default place this string data into constant data sections in the binary image.

Many binary file editors will display the ASCII interpretation of the raw hex data, so visually scanning the binary file and finding the address of each string is a simple matter. Once you know the addresses, you can search the executable code to see where they're used and draw some conclusions about what each section of code does.

One protection technique, then, is to encode text strings so they don't show up as readable ASCII in the binary image. (The encoding takes place outside of your application software, so that only the encoded representations appear in the binary image.) On a character-by-character basis, each character in the string can be mathematically manipulated in some way (for example, ANDing with a constant eight-bit "key") before being stored. The character must then be decoded before being sent to an output device. Or the encoding can take place on groups of characters.

In either case, you'll have to modify your language's output facilities, or perhaps write your own from scratch. In C, for instance, `printf()` has no inherent decryption capability, so you might choose to write your own version of `printf()` that knows how to turn an encoded string into legitimate ASCII data at run time.

If you are going to use string encryption, it's good practice to store critical information that is required to decode strings in the microprocessor's internal memory. For example, if you AND each character with an eight-bit key, the key should be stored within the microprocessor, so that it does not appear in the binary image anywhere.

Write your own operating system. If your application includes a commercial OS, certain "signatures" appear in

Countless individual internet appliances.



Imaging	Internet Appliances	Internet Infrastructure
Telecommunications	Intelligent I/O	Wireless
Aerospace/Defense	Automotive	Industrial Control

Whatever your market, we have your embedded software.

One common element.



 **WindRiver**
SYSTEMS

In the booming market of Internet appliances, success is all about time-to market, cost and product differentiation. If you design to win in that environment, there's one company with whom you'll definitely see eye to eye — **Wind River Systems**.

With our **Tornado**™ embedded development tools and **VxWorks**® real-time operating system, you'll have a huge head start in your product development cycle. Our open, reliable

architecture will allow you to differentiate your product for any sized design, all the way from the end-user application code to the look and feel of the graphical user interface.

By utilizing Wind River technology, Liberate Technologies (formerly known as NCI) rolled out its interactive TV software across Europe and the US, and TeraLogic, Inc. successfully launched its Cougar Reference Platform for HDTV. With over 200 Java design wins,

we've also made Java work in the embedded market. And our Universal Graphics Layer (UGL) provides a solid foundation for graphics development in the embedded space.

Find out more about Tornado and VxWorks. Visit www.wrs.com/html/espgene.html or call 1-800-545-WIND. Then give your designs the advantage of Wind River Systems. Because your competition is so fierce, you can't afford to blink first.

If the goal is to write well designed code for easier maintenance, then making poorly designed code in an attempt to protect it seems silly. However, critical algorithms may be obfuscated somewhat to hamper their decompilation and interpretation.

the binary image. The OS must include a start-up sequence, which is probably well documented by the OS vendor. Other run-time facilities will show up in the executable, and once these are found their interaction with your application code can be more quickly deciphered. And if the OS code is well designed and well coded, a decompiler program might have little trouble reconstructing the source code for OS functions.

If you use your own OS, a reverse engineer might need more time to figure out how your run-time environment works. However, writing your own run-time environment may not always be an option. Project schedules or legacy designs may dictate that you use a commercial OS. A home-grown kernel may only be a viable option for small, lower-memory projects that don't require accelerated design cycles.

Scramble address lines through extra logic. Another technique to hamper decompilation is to scramble address and/or data lines on your product's PC board(s). Instead of routing the address and data lines directly from the microprocessor to the ROM, you can insert some extra logic in between. An EPLD or FPGA can be a good source for such logic. So the address put out by the microprocessor, for example, will not be the physical ROM address to which an access is made. The ROM image will appear nonstandard, since the addresses have been essentially encoded. This fact alone might tip off a reverse engineer that something has been done to protect the code.

This method has other drawbacks, though. If your product can be instrumented with a logic analyzer, a reverse engineer can observe the "real" data being read by or written from the microprocessor. So while a decompiler

may have a hard time with the ROM image, instruction sequences can still be determined at run time. In addition, you have the overhead of designing around the address/data line scrambling logic yourself, which may require you to have some type of special utility to create your ROM-able image, since linkers generally won't support this type of operation.

Replace library functions. This method follows the same philosophy as using your own kernel. Since programming languages have standard libraries of functions, you could choose to not use these functions and instead write your own. This would make searching for common functions, such as `printf()`, more difficult. But this adds a significant overhead to your design cycle, since you have to reinvent the wheel.

Write lousy code. This option may seem ridiculous, but in fact might slow down the potential reverse engineer. The more convoluted a software design is, the more difficult it is to figure out, even at the source-code level (especially if it is also poorly documented). I'm certain there are a myriad of ways that code can be made to be "lousy," and I won't begin to offer suggestions for writing lousy code here.

I'm not really suggesting that this is a viable protection scheme, since it flies in the face of every software methodology written. If the goal is to write well designed code for easier maintenance, then making poorly designed code in an attempt to protect it seems silly. However, critical algorithms may be obfuscated somewhat to hamper their decompilation and interpretation.

Implement critical algorithms in silicon. This is perhaps one of the most realistic options for product protection. If your application contains proprietary

algorithms that are crucial to understanding how your product really works, implementing those algorithms in hardware rather than as code may be a good idea. Even if someone successfully decompiles your ROM, key pieces of the logic will be missing. For many applications, this may be enough protection to keep your product from being copied or modified.

Seal the electronics. The circuit board that includes your ROM could be encased in some sort of hard plastic sealant (potting or conformal coating) to make access to the device impossible. A would-be reverse engineer would have to physically destroy the product to get at the software. While this method may offer suitable protection against decompiling, it simultaneously presents a maintenance problem for you—the circuit board cannot be legitimately accessed to repair or replace the ROM and any other parts encased in sealant.

Use a home-grown microprocessor. If you have the time, resources, and need, designing your own microprocessor would provide a high level of protection against code decompilation. Assuming the architecture and instruction set for your processor are not publicly documented, the ROM image should be meaningless to anyone not familiar with your design. But this method is a large commitment, since you not only have to design the processor, you also have to create any compilers, assemblers, linkers, and debuggers to work with it.

However, an alternative does exist. Some microprocessor core companies provide licenses for their cores that include development tools to modify existing instructions, or add your own new instructions. The processor development tools allow you to generate a new core, plus they can generate a compiler for software development on your specialized processor. (Tensilica's Xtensa Processor Generator is one example of such a configurable processor environment.) As a result, you can create your own unique



Rational Rose for real-time?

The result could be painful.

**More and more development
teams agree Real-time Studio[™]
is the better system and
software modeling solution.**

It makes perfect sense. If you're developing real-time embedded systems, you need a software modeling solution that can support your entire team and will actually fit your existing process. Make no mistake — reach for Real-time Studio. This is the one and only software modeling solution with an object-based repository enabling complete, up-to-date, shared views of your system. Finally, your entire development team can communicate and collaborate — from start to product delivery. Developed exclusively for real-time systems, Real-time Studio is non-intrusive, flexible and features automatic code generation for C, C++, and Java. Today, it's the proven solution for hundreds of developers and development teams. And the list keeps growing. Get started on the fastest path to the right product. Visit www.artisansw.com/real for more information and a demo of Real-time Studio, plus you can register for our upcoming Real-time UML seminar.



Real-time Studio[™]
www.artisansw.com/real

No developer was actually harmed or risked injury in the making of this ad.
Rational Rose is a registered trademark of Rational Software Corporation.
Real-time Studio is a trademark of Artisan Software Tools, Inc.

Even if a hacker were willing to physically damage a unit to gain access to the binary executable, he would be unable to modify the code and still use the unit to obtain service.

instructions to implement critical algorithms which need special protection. Burn an FPGA with your modified core and you have a working prototype for software development. The final microprocessor can then be fabricated as an ASIC.

Two scenarios

Let's discuss two basic scenarios that might warrant the protection of a binary executable. In the first scenario, we must keep someone from modifying the code in a particular device so that features cannot be over-ridden or circumvented. Preventing this person from decompiling and understanding the code isn't necessary; rather, we're interested only in preventing modification of the executable. In the second scenario, we *do* need to thwart decompilation, to protect intellectual property rights in a competitive environment.

The first scenario is exemplified by various communications services, such as cable and satellite television or the caller ID telephone feature. In both cases the service provider supplies a customer with special equipment. In the cable TV case, the customer needs a cable TV converter box; with caller ID, the customer needs a display unit which indicates who is calling. The service provider often charges relatively little for the hardware itself, since it isn't the hardware that generates long-term revenue. The hardware simply serves as a gateway for a customer to access information (the cable program or the caller ID), and it is this information for which the customer must pay.

In these two cases, we are interested in protecting the firmware running in the customer's hardware from being altered in a way that enables the customer to get access to the service without paying for it. But it isn't nec-

essarily important to keep someone from decompiling the software to understand how it works—no revenue will be lost unless the software is modified in an attempt to circumvent billing.

The manufacturer of the cable converter or caller ID unit could employ the checksum technique to ensure that the embedded software cannot be altered and still operate. If packaging limitations allow, the electronics could be encased in sealant to make access to the ROM difficult or impossible without destroying the unit. Better yet, both methods can be used at the same time to offer increased protection. Even if a hacker were willing to physically damage a unit to gain access to the binary executable, he would be unable to modify the code and still use the unit to obtain service.

The second scenario leans more toward the protection of intellectual property and the threat of bona fide competition in the marketplace, not just casual hacking. Any product, particularly those that are cutting-edge in their specific industries, can be a target of reverse engineering by an unethical competitor. In these cases it may be worth the cost to implement some of the more aggressive protection techniques listed in the survey. In particular, the home-grown microprocessor technique may prove to be a viable option.

Consider a hypothetical example: let's say your company has developed a new type of portable medical diagnostic device that allows patients to monitor some aspect of their own health in real time (this is a hypothetical example, remember). Your company is pursuing a patent on the technology, but you need to get the device to market as quickly as possible. In this case, especially if the potential market is large, you may decide to modify an

existing processor core to hide key aspects of your algorithms. In this way potential competitors will not be able to unlock the secrets of your application unless they first get information about your specialized processor and the instruction set it uses.

Weighing the costs

The code protection techniques that aren't too costly to implement can be somewhat readily overcome by competitors, while those techniques that are more iron-clad may require a substantial investment. It may be that the cost to implement such an iron-clad method overwhelms the "normal" cost of development for the product. The bottom line is, if someone is willing to spend the time and money, little can be done to stop reverse engineering of embedded code. Even if you use custom hardware, there are clever engineers who understand a commercially available CPU well enough to make logical (and often correct) assumptions about the details of your design.

A custom microprocessor with an unpublished instruction set, or physically sealing the electronics in some manner, might satisfactorily protect the software, but these may not be viable solutions for many applications (particularly in smaller organizations with smaller development budgets). Ultimately, the decision of whether or not to put protection schemes in place may in fact be a business decision and not a purely technical one. The company must weigh the cost of implementing code protection vs. the potential risk of intellectual property infringements or loss of service revenue, and decide what is best for its particular situation.

esp

Matt Fisher is a software engineer who has been working with embedded systems for the past five years. He has experience in the defense, telecommunications, and electronics industries, and holds a BS and MEng in electrical engineering from Rensselaer Polytechnic Institute in Troy, NY. He can be reached at matt_fisher@ieee.org.



Introducing...

DSP Enabling Technologies™

- Development Hardware
- Operating Systems
- Bundled Toolsets
- Design Services
- Consulting

Blackhawk

by EWA Inc.



Phone: 1-877-983-4514

For more information please visit: www.blackhawk-dsp.com

Unit Testing Prevents Errors

by Adam Kolawa

In previous columns, we have focused on coding standards as a type of error prevention. This month we shall introduce unit testing, which is another important facet of the error prevention process. To understand the importance of unit testing, we must first take a look at the current state of software testing.

Developers today face increasing deadline pressure as they write code in a competitive software market. They often write code quickly and send it off to another department for testing. However, the best person to test code is the developer who wrote it. He or she understands the code better than anyone else and is most capable of finding the weaknesses in the code.

As developers test code, they should not just be trying to finish quickly and go home early. Rather, they should be trying to break the code in any way possible. Instead of mentally rewarding yourself for code that appears clean, you should reward yourself for finding bugs. Adopt a new, more critical outlook on your code, and you will see dividends in the cleanliness of that code.

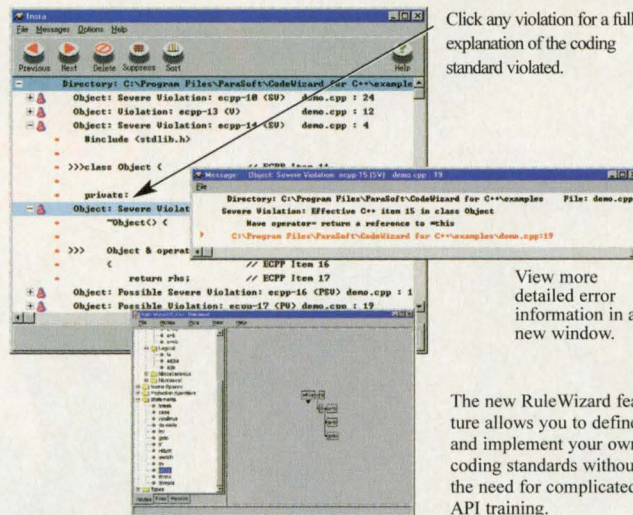
Once you have adopted this new outlook, you need to find an effective method of testing. Unit testing is one of the most effective methods available.

Developers often hear about unit testing and think the term refers to module testing. But unit testing actually refers to an even lower level of testing that occurs at the class or object level. Class-level testing is extremely important to the well-being of any application. However, most developers avoid class testing and believe that it is not possible to perform on large applications because of the labor and techniques involved.

In upcoming columns, we shall illustrate that although manual unit testing at the class level involves some extra work, the technique is certainly feasible and worthwhile. Furthermore, technology has evolved to the point that developers can now choose from a number of powerful tools that make unit testing as easy as breathing.

Dr. Adam Kolawa is President and CEO of ParaSoft. You can reach him by e-mail at ukola@parasoft.com.

Advertisement



Click any violation for a full explanation of the coding standard violated.

View more detailed error information in a new window.

The new RuleWizard feature allows you to define and implement your own coding standards without the need for complicated API training.

Would you believe that CodeWizard® can prevent errors in your code and cut your debugging by 25%?

"CodeWizard is the code reviewer without the attitude; it knows all the rules." -- Suzette LaGray Siemens ElectroCom L.P.

CodeWizard® is a tool that helps you prevent errors by automatically performing static analysis on C++ code. The first thing you need to know about CodeWizard is that it can help you do less of the thing you hate the most: debugging. With CodeWizard, preventing errors before they happen is painless.

CodeWizard automatically enforces over 70 C++ coding standards, which are language-specific rules that help you to avoid dangerous coding constructs. The standards are borrowed from the writings of Scott Meyers and other industry experts. CodeWizard clearly displays each of your violations and gives you all the information you need to fix them, including suggestions on better coding constructs. No more wondering about weaknesses in your code. Follow CodeWizard's coding suggestions, and you'll immediately begin to write cleaner, more reliable code.

The key to CodeWizard is that you can configure the program to avoid overworking yourself. With a few clicks, you can tell CodeWizard to enforce

only the standards that are most relevant to the current project. Using the new RuleWizard™ feature, you can even write your own coding standards with point-and-click convenience. Use CodeWizard and RuleWizard together, and you'll have a personalized tool for your entire development team. This is a tool that conforms to you—not the other way around.

You won't have any problem adding CodeWizard to your arsenal of development tools, because it installs directly into Windows® and UNIX development environments. In Microsoft® Developer Studio®, you'll begin testing with a single click of the CodeWizard icon. On UNIX platforms, CodeWizard is a wrapper around the compiler. Learn a few simple commands, and you'll be on your way to cleaner code.

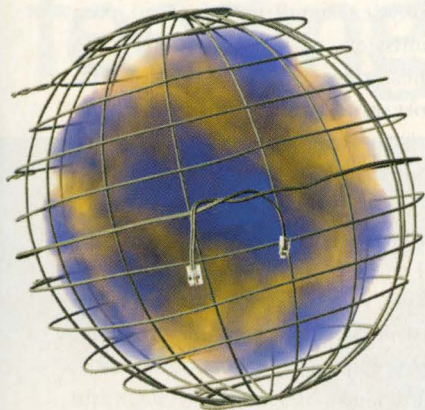
If you would like to prevent errors, reduce debugging, and go home on time for a change, download a fully-functional demo copy of CodeWizard today at www.parasoft.com/best.htm, or call (888)305-0041 for more information.



www.parasoft.com/best.htm



Michael Barr



Virtual Serial Ports

Serial ports are extremely versatile peripherals. They can be used for communication with external equipment, to provide low-cost network connectivity, for data-logging purposes, and in myriad other ways. As a result, finding several serial ports on each piece of embedded hardware that you work with isn't uncommon. But what if you need a serial port when there are none, or all of them are already being used for other purposes? This month, I'll show you how to create virtual serial ports from other hardware that you may already have at your disposal.

But first, let me tell you about this month's Internet Appliance Design section. The feature articles treat two very different subjects. The first, by David Crawford and Emmanuel Roy, is about speech coding techniques. It focuses primarily on the implementation details you're likely to encounter if your application involves speech coding on a digital signal processor.

The second article, by Scott Lawrence, is about the eXtensible Markup Language (XML). In the process of illustrating some uses for XML in embedded systems, comparisons are made between XML and its better-known cousin, HTML. XML, it seems, is more useful for computer-to-computer information exchange, particularly in large distributed systems such as an enterprise network.

A serious omission

A couple of years back, I joined a project for which the hardware had already been designed and the first

prototypes manufactured. It was my job to design and implement the software for the embedded processor on board. This all sounded great. I'd done embedded software design and development work several times before and the particulars of this project's application were exciting to me.

However, after looking at the schematic for the hardware and talking with the board's principal designer, I quickly noted a serious omission: there were no serial ports. Nor was there any room to add them—this nine-layer, full-width PCI card was already populated to the max with processor, peripherals, expansion buses, FPGAs, and memory. Serial ports were not strictly necessary for the initial application, so they hadn't even been considered. But I could already think of two immediate uses for serial ports and at least one potential use for another down the road.

Here's how I would've liked to use serial ports on this board:

- **Debugging.** A debug port would allow me to use an off-the-shelf remote debugger to step through my firmware, set breakpoints, and watch variables and the stack during the development phase. A serial port would be the best physical interface, since the majority of remote debuggers expect that type of connection to the target processor
- **Data-logging.** The ability to dump text strings would make the behavior of the system in the field much easier to understand and improve. A serial port would be ideal, since I

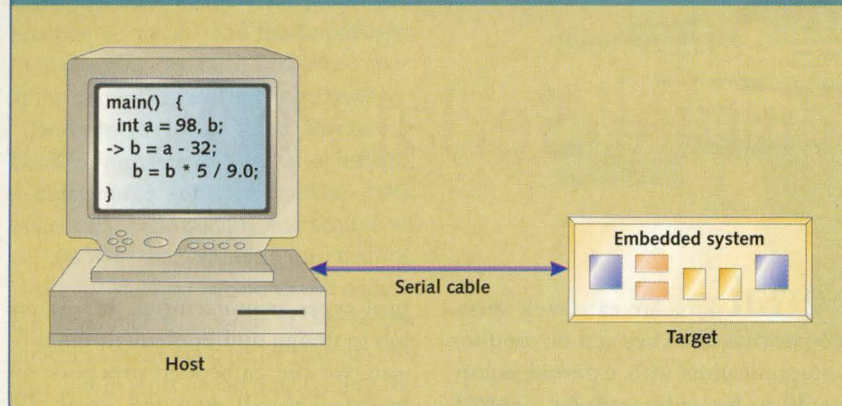
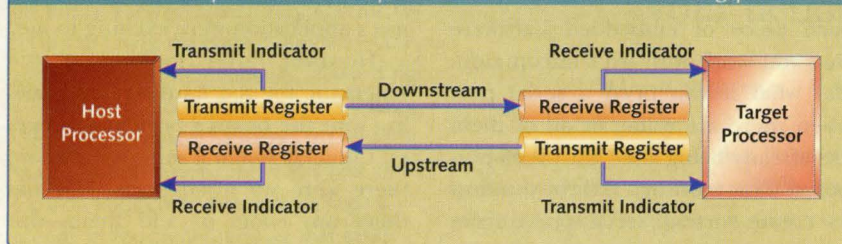
Internet Appliance Design

in this issue

39 Implementing Speech Coding Algorithms

55 Embedding with XML

61 Embedded Internet Tools

FIGURE 1 Traditional serial communications**FIGURE 2** The parts of a serial port visible to the communicating processor

could easily modify the stdio facilities of my standard C library to write data to a serial port via `printf()` and read it in via `scanf()`

- **Interprocessor communication.** This product could potentially be used as a coprocessor, rather than just as the standalone I/O processor it was initially intended to be. For that potential to be realized, there'd have to be a communications channel between the on-board embedded processor and the microprocessor on the motherboard. A serial port would be one way to provide the basis for that future capability

Faced with this need for serial ports which were simply not available, I was forced to improvise. The solution I came up with was to simulate a set of four serial ports over the PCI bus. This gave our product the ability to perform debugging and data-logging immediately and a way to add interprocessor communication and any

other serial application a customer might dream up in the future. All of these communications would take place between the embedded processor on our PCI card and the host processor in the Sun workstation or PC in which the card was installed. Since all of these communications would be over the PCI bus, this solution would require no actual physical serial ports from the host workstation or PC—an unexpected side benefit of virtualization.

Serial communication, in the abstract

Serial data transmission is traditionally accomplished with two processors, two serial controllers, and a section of cable between them. Figure 1 shows what this looks like in the case of a typical remote debugging scenario. Two data channels lie within the serial cable: one for upstream (toward the host) data and one for downstream (toward the target) data.

For virtualization purposes, that's all we really need to understand about the physical medium of serial data transmission.

These days, the basic unit of data in most serial applications is one byte. Each byte of serial data is transmitted one bit at a time, with its eight bits framed by start and stop bits and an optional parity bit. All of these bits travel within one channel of the serial cable, reaching the other end of the cable shortly after they're sent. Upon receipt, the serial controller at the other end strips off the start and stop bits, collects the eight data bits into a byte, confirms the parity bit is correct (if parity is enabled), and places the incoming data into a byte-wide receive register that is readable by the processor.

The processor on the receiving end sees only the byte in the serial controller's receive register. It does not know, or even care, about the rest of these details. The same is true during transmission: the sending processor simply writes the byte of data it wants to send into the local serial controller's transmit register. The start, stop, and parity bits are added by the serial controller and are necessary only because of the asynchronous nature of the transmission and the potential for noise on the physical channel.

To each of the processors involved in the communication, a serial data channel looks like nothing more than a pair of registers and a pair of indicators:

- The *transmit register* contains the next byte of data to be transmitted
- The *receive register* contains the last byte of data received
- A *transmit indicator* indicates that the data in the transmit register has been sent
- A *receive indicator* indicates that a new byte of data has been received

Figure 2 is a pictorial representation of the processor-visible aspects of a serial port.

Windows CE is feeding cows.

Powered by
Microsoft
Windows CE



m&f
Engineering Team

When you've got hundreds of hungry bovines to feed, you don't have time to think about whether your operating system can run entirely in CompactFlash. Or if it supports the Microsoft® Win32® API. But luckily for busy dairy farmers, Förster Technik does. And that's why, with the assistance of Mettler and Fuchs AG, they powered their TeamMaster industrial automatic nursing station with Microsoft Windows® CE.

Windows CE brings the ease-of-use of a PC to rugged embedded devices—in any industry. With it, dairy farmers can experience the familiar Windows environment from their animal's stalls, feed calves in the field and monitor the herd's health in the office—all while streamlining their business.

Windows CE and the powerful Win32 API facilitate code reuse and feature expansion, so Förster Technik can add modules to the TeamMaster to dispense feed and weigh cows. And the robust Windows CE operating system can run entirely from RAM and ROM or CompactFlash, making it the perfect RTOS for ruggedized devices.

Dairy farmers don't design embedded systems. But you do. So why not use the environment that's familiar to both users and developers?

Win the race to market. Power your next generation of intelligent devices with Windows CE.

Visit www.microsoft.com/windowsce/embedded

Where do you want to go today? **Microsoft**

Of course, most serial controller chips also have additional registers (for controlling the data rate and other properties of the physical inter-

face) and additional interrupts or flag bits (to indicate various communication errors). However, none of these is strictly necessary.

In particular, it's important to note that the serial data rate (for example, 9600 bits per second) is not of concern to the communicating processors. The serial data rate merely establishes that the same clock is used by the two serial controllers when shifting bits into and out of the physical channel. The maximum data rate under the scheme I'm about to describe is subject only to the limitations of the processors and the hardware used to implement the virtual serial ports.

Virtualizing the serial port

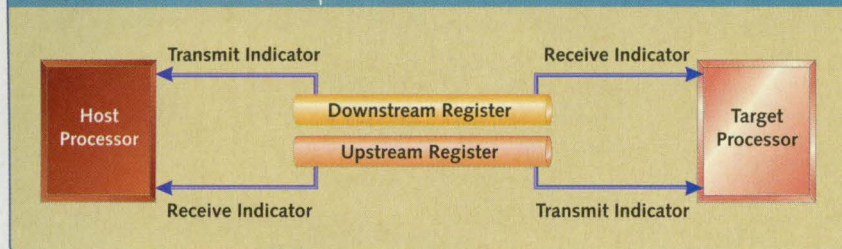
No serial controllers or cables are required to implement a virtual serial port. All you need is a pair of byte-wide registers that can be read and written by both of the communicating processors and some way to set and clear the two indicators. In fact, the design of a virtual serial port is actually much simpler than a physical serial port because no potential exists for noise or loss of information on the channel. The transmitting processor simply places the data into a register and the receiving processor takes the data out of the same location. This is exactly how a properly functioning serial port with zero delay would behave. Figure 3 illustrates the setup.

When a serial port is working properly, we can imagine that the host processor's transmit register is directly connected to the target processor's receive register. Any data written into the host's transmit register will appear in the target's receive register, after a brief delay. As this "brief delay" approaches zero, the two registers may be thought of as one and the same physical location. This observation applies in both directions of transmission.

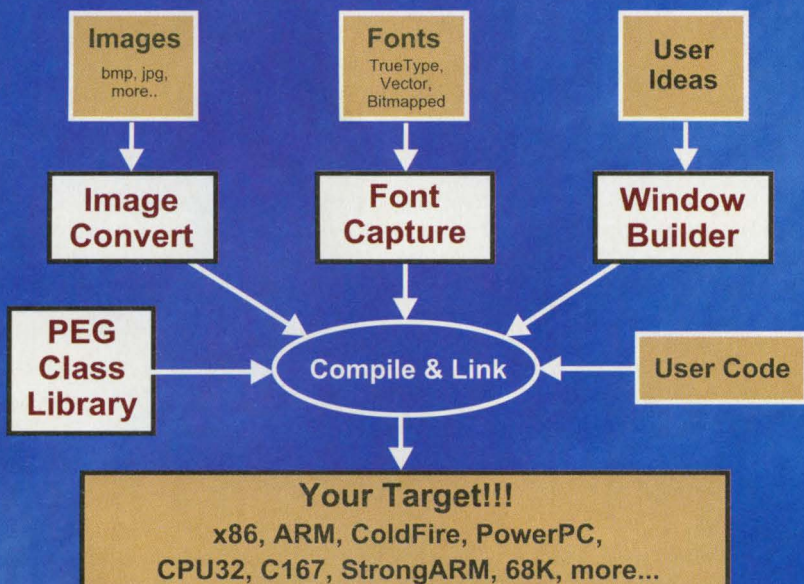
The two registers of a virtual serial port are referred to as the *upstream* and *downstream* registers. This eliminates any potential confusion between the names *receive* and *transmit* (each processor sees the register as serving a different purpose in that regard). For compatibility with existing application software that communicates over a serial port, these registers should always be eight bits wide.

The key to communicating like this is proper synchronization between the processors. We must ensure that the host processor never overwrites data that was unread by the target processor, and vice versa. The actual synchronization technique involves the use of four indicators. Each receive indicator (interrupt or flag) indicates that the

FIGURE 3 A virtual serial port



PEG™ does this...



Portable Embedded GUI

www.swellsoftware.com

Call for a FREE Eval Kit 1-800-366-2491

other processor has just transmitted a byte. Each transmit indicator (interrupt or flag) indicates that the other processor has just received a byte.

The following symmetrical algorithms can be used to send and receive data over a virtual serial port. Through a creative implementation of the initialization code, it may even be possible to use the very same receive and transmit function implementations on both processors. When the algorithm says to wait on an indicator, the processor can either poll the appropriate flag or have an interrupt handler respond to an interrupt asynchronously.

Transmit algorithm:

1. Wait for the local transmit indicator to be set
2. Write the next byte of data into the transmit register
3. Clear the local transmit indicator
4. Set the remote receive indicator

Receive algorithm:

1. Wait for the local receive indicator to be set
2. Read the byte of data from the receive register
3. Clear the local receive indicator
4. Set the remote transmit indicator

That's pretty much all there is to it. To create a virtual serial port, simply:

- Select two shared memory locations or registers that are readable and writable by each processor. These are the upstream and downstream data registers
- Figure out what interrupts or flags you'll use for the four indicators
- Implement the two algorithms above (and perhaps an initialization function, as well) on both processors

You'll even be able to test your application software changes (if any) on a PC first, simply by writing a stub that performs the virtual serial port function via memory shared between two threads or applications.

No serial controllers or cables are required to implement a virtual serial port. All you need is a pair of byte-wide registers that can be read and written by both of the communicating processors and some way to set and clear the two indicators.

Implementation details

In my particular situation, I found the perfect location for my upstream and downstream registers within the PCI bridge chip on our PCI card. The particular chip we were using, the PLX9080, happens to have eight 32-bit "mailbox registers," which can be read and written from both sides of the PCI bus. That is, the microprocessor on the motherboard (a SPARC or a Pentium, for example) and the embedded processor on the PCI card could both access any of these eight registers. For simplicity's sake, I used one of these 32-bit registers for each of the eight-bit registers in the virtual serial port. This made addressing the mailbox registers simple. Since there were eight mailbox registers and only two registers are necessary to implement each virtual serial port, I was able to implement a set of four identical ports within that one PCI bridge chip.

The receive and transmit interrupts were implemented within the "doorbell registers" of the same PCI bridge chip. Access to these two 32-bit registers is more restrictive than the mailbox registers. In my case, I designated one upstream doorbell register and one downstream. The bits in the "upstream doorbell" could only be set by the embedded processor on the PCI card and cleared by the host processor. When the embedded processor set one of the doorbell register bits, the PCI bridge chip would generate an interrupt to the host processor, so that the host processor would note the interrupt, take the appropriate action, and clear the relevant bit in the doorbell register. Downstream interrupts are generated and cleared in the reverse manner. Since each virtual serial port requires two interrupts in each direction, a

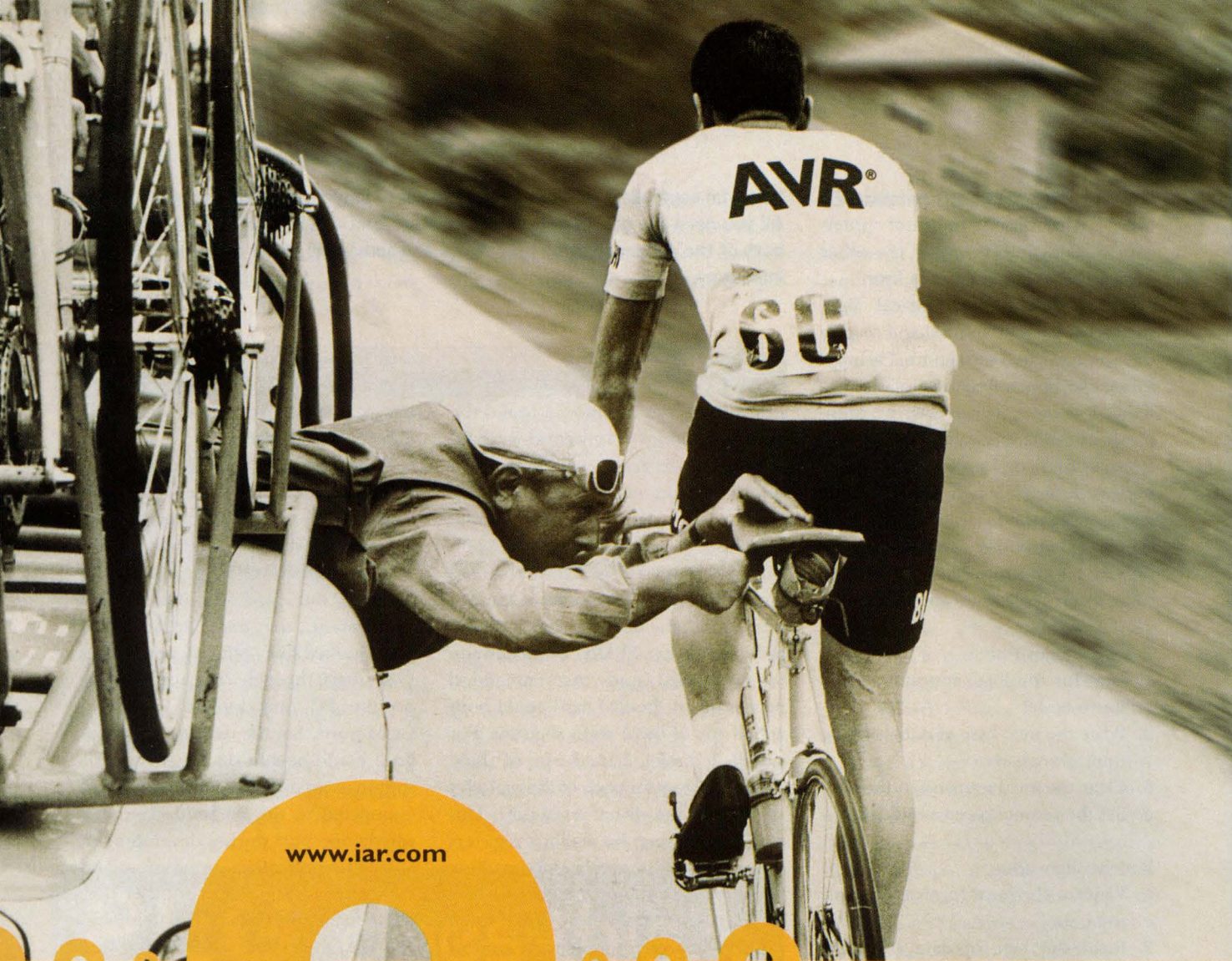
total of eight bits from each doorbell register were involved. The other 24 bits of each register remained available for other uses.

The software that implemented the host part of the virtual serial port functionality was made part of the Solaris and Windows NT device drivers we shipped with our product. To users of our card who had a Sun workstation, the four virtual serial ports looked like ordinary device files (`/dev/vsp0` through `/dev/vsp3`), while on the PC they looked like extra COM ports. So, the debugger (I used `gdb`) could be run on the host workstation as an ordinary application and connected to the embedded processor via one port during development; a small application program could capture text dumped to another port while the system was in the field; and the way was made clear for future applications to use up to two additional virtual serial ports.

I hope you can find use for this technique in your own work. I can't post the source code, since it is owned by my previous employer. However, I may be able to help if you get stuck on the implementation. Next month I'll be talking about the potential implications of inappropriate programming language choice. In the meantime, stay connected...

esp

Michael Barr is the technical editor of Embedded Systems Programming. He holds BS and MS degrees in electrical engineering from the University of Maryland. Prior to joining the magazine, Michael spent half a decade developing embedded software and device drivers. He is also the author of the book Programming Embedded Systems in C and C++ (O'Reilly & Associates). Michael can be reached via e-mail at mbarr@mfi.com.



www.iar.com

IAR and Atmel – fast moving microcontrollers with professional tools support

Atmel's AVR 8-bit RISC microcontrollers are on the move. Fast and competitively priced, they are rapidly winning ground with embedded developers for small applications as well as those demanding high performance tasks. IAR Systems played a key roll in this success. The AVR is well suited for high level language programming and the IAR Embedded Workbench now offers the choice of C/EC++ to most AVR applications.

Today, IAR continues to cooperate closely with Atmel, and provides a complete development solution:

- **IAR Embedded Workbench™ for AVR**, a state-of-the-art integrated development environment with a highly optimized C/EC++ compiler
- **IAR MakeApp™ for AVR**, a device driver wizard generating all initialization code
- **IAR visualSTATE® for AVR**, a graphical development tool with a patented technology to generate production ready code

Clearly, a unique solution that is hard to beat.

As IAR supports more than 30 different architectures, you are probably already familiar with IAR's development environment. This familiarity will help reduce start-up time for your next AVR project. Therefore if you want to reach your goals quickly and without costly interruptions, make sure to adopt the IAR solutions for your embedded systems development. When you're racing against the clock, there's no better partnership to have at your side.



IAR and Atmel – embedded development on the move

For more information, contact IAR at 1-800-427-8868

OR email: info@iar.com



DIFFERENT ARCHITECTURES.
ONE SOLUTION.

USA & Canada 415 765 5500, info@iar.com Germany +49 89 90 06 90 80, info@iar.de
UK +44 171 924 3334, info@iarsys.co.uk Sweden +46 18 16 78 00, info@iar.se Denmark +45 8625 1111, info@iar.dk
ATMEL AND AVR ARE REGISTERED TRADEMARKS OF ATMEL CORP. EMBEDDED WORKBENCH, MAKEAPP, VISUALSTATE ARE TRADEMARKS OF IAR SYSTEMS.

Implementing Speech Coding Algorithms

Here are some techniques you can use to implement speech coding standards in a real-time DSP system.

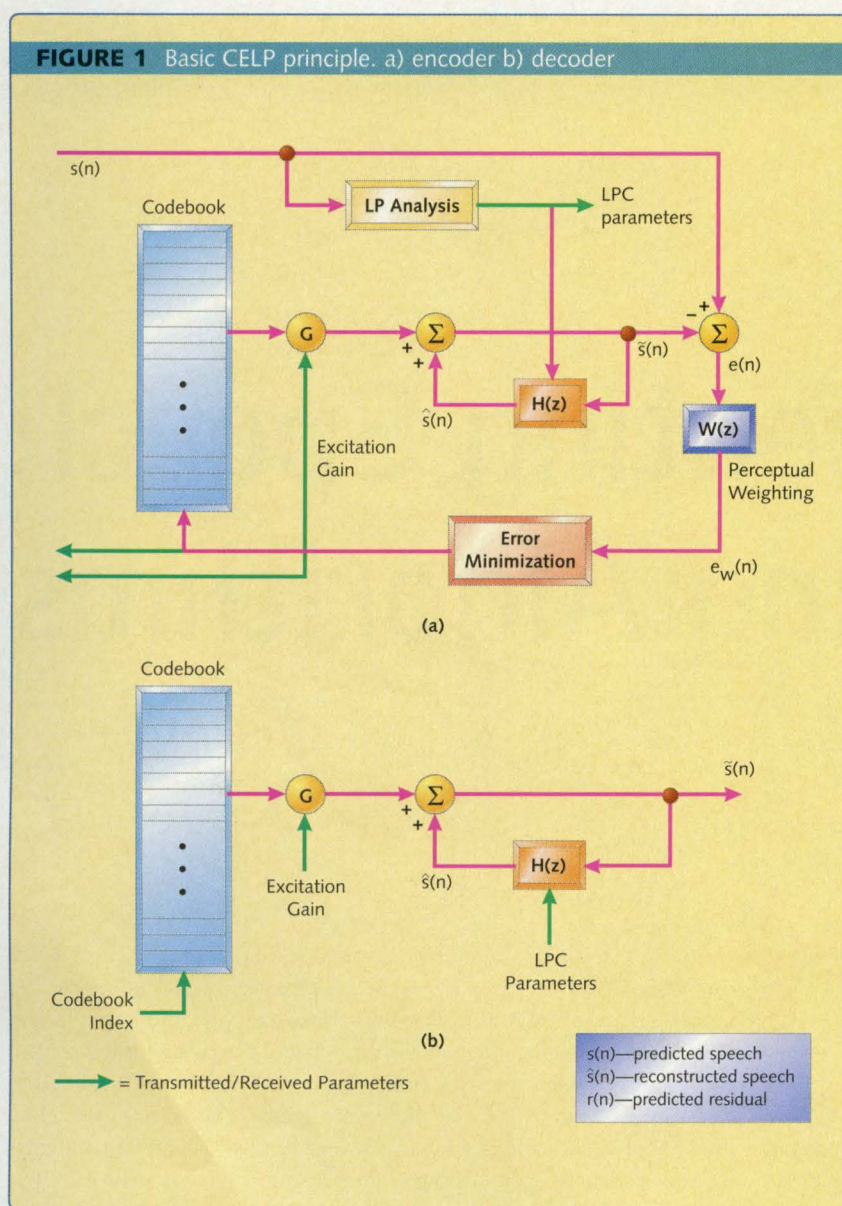
Over the last few years, engineers have spent a great deal of effort developing methods of coding digital speech signals for applications such as mobile communications, transmission of speech over the Internet, and mass storage by call centers of recorded telephone conversations. Such coding involves the removal of redundant and irrelevant information in the speech signal and can lead to reasonably good-quality speech at bit rates lower than 8kbits/s. However, to achieve lower bit rates while maintaining an acceptable quality of speech, speech coding algorithms are becoming increasingly sophisticated and more demanding in terms of the number of calculations required per second. Therefore, a need exists for efficient implementation of speech coding algorithms on digital signal processors (DSPs). This article describes techniques and approaches commonly used to realize such implementations.

Speech coding standards

To achieve efficient interworking between different telecommunication networks, standardization of speech coding algorithms is necessary. Organizations such as the International Telecommunication Union (ITU), the European Telecommunications Standards Institute (ETSI), and the Telecommunications Industry Association (TIA) act as standards bodies, defining and publishing telecommunications standards.

In general, the items available from the standards bodies include:

- Documentation describing the algorithm and references to related standards and technical publications
- ANSI C code corresponding to a fixed-point fractional arithmetic implementation of the codec (hereafter referred to as the *standard reference*)

FIGURE 1 Basic CELP principle. a) encoder b) decoder

- A set of test vectors, which comprises input and output signals that can be used to verify code being developed

In this article, we'll describe the development of real-time speech codecs on a DSP. We'll draw examples from the GSM Enhanced Full Rate (EFR) codec.¹ However, the techniques are applicable to almost any type of speech codec.

Speech coding techniques

Speech coding has made remarkable progress over just a few decades. One of

the earliest and most widely used techniques in commercial telephony is A-law/ μ -law PCM, in which each 13- or 14-bit speech sample is quantized to eight bits using a non-uniform quantization characteristic.² This technique has the effect of allowing a sufficient dynamic range to be covered using fewer bits than with linear quantization, while maintaining a reasonably good signal-to-noise ratio (SNR). For a sampling frequency of 8kHz, the bit rate is reduced from 104kbits/s or 112kbits/s to 64kbits/s.

The research community has addressed the field of speech coding

in an effort to further reduce the required bit rate. This effort has resulted in the emergence of a large number of speech coding algorithms of various types. Linear Predictive Coding (LPC) with adaptive prediction and adaptive quantization allows the bit rate to be reduced to about 32kbits/s while still maintaining good quality speech. Further reductions are possible through the use of Code-Excited Linear Prediction (CELP)³, in which the predictor's excitation is derived from a codebook of stored vectors rather than from the LPC residual signal, as illustrated in Figure 1. (The bit rate reduction arises from the fact that the LPC residual is not transmitted.) Bit rates down to about 16kbits/s can be achieved using CELP, and if speech quality is not of particular importance (for example, in military systems), bit rates below 8kbits/s can be used.

Algebraic code-excited linear prediction (ACELP)

One of the most popular techniques in current practical use is algebraic code-excited linear prediction (ACELP), which is essentially a development of the CELP technique. Although the CELP algorithm enables communication-quality speech to be transmitted at relatively low bit rates, it requires considerable computational effort to perform the codebook search and requires large amounts of memory for storing the codebooks. These issues are addressed by ACELP, which doesn't require explicit storage of codebooks. In ACELP, the encoder "builds" each codebook vector according to a set of rules and constraints governing the positioning of pulses within the vector. It then selects the optimal vector and transmits the positions and amplitudes of the pulses to the decoder. The decoder uses this information to reconstruct the excitation vector by placing the pulses in their correct positions rather than indexing into a stored codebook. The algebraic codebook structure also

HEY!

We're in here!

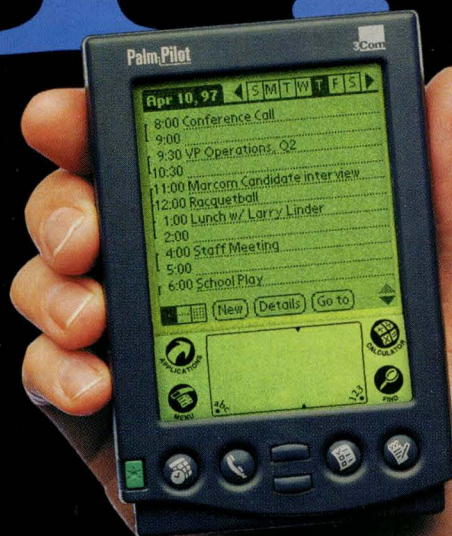


AMX™

Real-Time Multitasking Kernels

KADAK

AMX is a trademark of KADAK Products Ltd. 3Com and the 3Com logo are registered trademarks and PalmPilot, and the PalmPilot logo are trademarks of 3Com Corporation or its subsidiaries. All other trademarked names are the property of their respective owners.



We've stayed
out of sight for far
too long.

It's time everyone knew ...

that KADAK has been

quietly setting real-time standards since 1978.

That's right, from Sony to Hewlett Packard, Philips to Hughes Aircraft, over 1,000 of the most demanding companies in the world have chosen our AMX™ kernels to provide the processing power for all manner of smart devices.

From hidden embedded devices to the market-leading 3Com PalmPilot™ connected organizer, they've come to count on KADAK for superior real-time multitasking kernels, crystal-clear documentation, highly-responsive technical support and the ultimate in work and time-saving tools and utilities.

So, next time you need a real-time multitasking kernel remember ... You can count on **KADAK!**

Visit us at www.kadak.com



allows fast search procedures to be employed, since each code vector is limited to a finite number of pulses.

Many of the recent speech coding standards, including GSM EFR and G.729, have used the ACELP principle.^{4,5} Details vary from one standard to another. For example, the GSM EFR standard allows 10 pulses per 5ms subframe, with certain restrictions on which positions each pulse can occupy. The G.729 standard, on the other hand, has only four pulses per 5ms subframe.

Reference code

Before embarking on a real-time implementation of a speech codec on a target DSP, it is advisable to spend some time and effort developing a working reference implementation of the codec in a high-level language (such as C) which runs on a standard PC/workstation. Unlike the standard

reference code obtained from the standards body, the working reference code will change as the development progresses. Each change that is made to the DSP code should be reflected in the working reference code. If, for example, it is decided that a function will be in-lined rather than called, the working reference running on the PC/workstation should also be modified in this way. As the development progresses, the working reference code will deviate more and more from the standard reference code (although its function should remain identical, and it should still pass all test vectors). The benefits of having a working reference running on a PC/workstation cannot be understated. The time and effort spent will prove itself well worthwhile when it comes to debugging the DSP code, especially if the piece of DSP code being debugged is written in assembly

language. The working reference will essentially be a C code version of the assembly language code.

Implementation on a DSP

When you start to implement the codec on the target DSP, your first decision is how to code the software. Three choices exist, each with its own advantages and disadvantages:

- Implement all code in C (or some other high-level language). This approach may be the easiest in terms of development time and effort. However, the resulting implementation is unlikely to be optimal, even with a good C compiler. If the processing power and memory are not of great concern, or if the codec will not be required to work in real time, this may well be the best approach to take. But in most cases, it will simply not give the performance you need for the application
- Implement all code in assembly language. This approach will, with sufficient knowledge of the DSP and its assembly language, lead to the most optimal implementation. However, the development time and effort will be significant. Furthermore, debugging and updating of the code will be more difficult, and the need for a good working reference model will therefore be greater
- Implement the codec using a mixture of C and DSP assembly language. This approach is arguably the most practical compromise between performance and development effort, and is the approach that will be adopted for the remainder of this article. Typically, the most compute-intensive routines in the codec would be implemented in assembly language, while the main body of the code would be implemented in C. At some point, the desired performance will be reached and the development will be frozen

Your RTOS Should Be *LEAN* and *MEAN*...



CMX-RTX™: Fastest Context Switch Time,
Lowest Interrupt Latency and Smallest Footprint.

• TCP/IP for 16-/32-bit CPU's

• No Royalties • Free Source Code

CALL TODAY FOR FREE WHITE PAPER:

"RTOS: Buy or Roll Your Own?"

CMX
COMPANY

8051
8051-XA
80C251
80196/296
80x86
80C16x
HC08/11/12
H8/H8S
68K/683xx
TLCS-900
AVR
M16C
SH
PowerPC
SHARC
TI DSP's
Many More!

508.872.7675

cmx@cmx.com

http://www.cmx.com

In broad terms, an implementation strategy should resemble the following:

1. Develop a working reference C code model for running on a PC/workstation. Make sure this code passes all test vectors
2. Starting with a copy of the working reference C code, make modifications to take account of specific DSP compiler and development system requirements that are non-ANSI (for example, file I/O on the DSP development system, usage of program memory and data memory, and so on). One possible approach is to use only one version of C code, with an `#ifdef DSP_CODE` preprocessor statement for differentiating between those parts of the code that are associated with the PC/workstation and those that are associated with the target DSP development system. (`DSP_CODE` should be defined when building the code for the target DSP, and should not be defined when building the code for the PC/workstation)
3. Compile the DSP code for the target DSP. If the resulting code fits within the DSP's addressable range, run it on the DSP simulator or on the DSP development hardware. The code will almost certainly consume a large number of MIPS, but this should not be an issue because the code will be operating in non-real time at this stage. If the code runs and passes the test vectors, the first stage is complete. Otherwise, as is more likely to be the case, debugging will be required. This is where the effort spent in developing the working reference C code for the PC/workstation will start to prove its worth. If the code doesn't fit within the DSP's addressable range, a smaller block will have to be extracted as a temporary measure. Once the code has been optimized enough to produce smaller code size, the smaller blocks can be

integrated again. A good approach in this case would be to implement only the decoder, since this will require a significantly smaller amount of memory than the encoder. The optimizations applied to the decoder can then be applied to the encoder, thus reduc-

ing the encoder code size. Alternatively, a small block of code can be extracted and implemented on the target DSP, with intermediate test vectors derived from the working reference C code on the PC/workstation

4. Having established that the code



It's faster — better. It's the new

RABBIT 2000™

MICROPROCESSOR FOR THE NEXT MILLENNIUM

This high-performance 8-bit microprocessor is loaded with on-chip peripherals. It runs 3X faster than its predecessors the HD64180 and Z180. Its fast number crunching outperforms most 16-bit chips. Powerful software support is provided by Dynamic C® an interactive compiler, editor and debugger.

- *Glueless interfacing*
- *1 Megabyte of code space*
- *4 serial ports*
- *Remote cold boot*
- *40-plus I/O pins*
- *Slave port*
- *"Sleepy" mode allows operation at 100-200 μ A*
- *7 timers, battery backable time/date clock*



Introductory Offer!
Rabbit 2000™ Development Kit
Only \$99

Limited offer ends March 31, 2000

Order online @
www.rabbitsemiconductor.com



2932 Spafford Street, Davis, CA 95616
Tel 530.757.8400 • Fax 530.757.8402

Includes:

Single-board computer with Rabbit 2000™ processor, prototyping board, Dynamic C® development software and documentation on CD ROM, power supply, and PC serial cable to perform real-time debugging.

*Dynamic C® is trademark of Z-World, Inc.

now runs on the DSP development system, the task of reducing the number of MIPS and amount of program memory can begin. This will typically involve modifying the C code to allow the compiler to make better optimizations, as well as replacing some of the key functions with versions written in assembly language

Optimizing the C code

In recent years, C compilers for DSPs have improved significantly, and most now have optimizing capabilities that allow them to generate more efficient assembly language code than was previously possible. Before any optimization begins, however, establishing which routines are the best candidates for optimization will be necessary. Little can be gained by optimizing a routine if it was already close to optimal. To determine whether a routine could benefit from further optimization, two pieces of information are required:

- The number of cycles the routine is expected to consume. This number can be obtained by performing a weighted million operations per second (WMOPS) count. The facility to do this exists in the standard reference code and should also be implemented in the working reference code. A C structure maintains a series of counters, each associated with one of the main operations in the code. Each time an operation is executed, the appropriate counter is incremented by an estimate of the number of cycles that operation will take on a typical DSP. This allows the developer to obtain an estimate of the optimal performance, and while not entirely accurate, it will certainly give a good idea of what the performance target for a particular routine should be
- The number of cycles currently consumed by the routine over a frame. (This will be proportional to the number of cycles taken to exe-

cute the routine and the number of times this routine is called during a frame.) To obtain this figure, the DSP development tools suite will usually have a profiler. The profiler should allow the developer to obtain such information as the number of calls made to the routine, the number of cycles taken by each call, the total number of cycles consumed by all calls, and so on. Note that the worst case figure for a whole frame should be obtained

Therefore, if a certain routine currently requires, say, 1,000 cycles, and its WMOPS figure is 50, then this routine would be an ideal candidate for further optimization. If, on the other hand, the WMOPS figure was, say, 950, then the current 1,000 cycles is already fairly close, further optimization may yield little gain. Note, however, that some small functions are called many times in, for example, an inner loop, and saving even just one cycle in such a function can have a significant effect on the overall computational requirements.

Typical optimization issues you need to take into account include:

Memory partitioning. Most DSPs have a mechanism by which two data operands can be loaded from or stored to memory in a single cycle (for example, X and Y data memory in the case of the Motorola DSP56300 family⁶), so that parallel data loads/stores can take place. Significant processing power can often be saved if a variable is stored in, say, Y memory instead of X memory, since the opportunity then exists for moves to/from Y memory in parallel with moves to X memory, rather than having to perform two separate moves from X memory. As an example, the GSM EFR function `Residu()` might be modified such that the coefficients are expected to be in Y data memory. This would lead to the following prototype (assuming you're using TASKING's C compiler for the DSP56300 family⁷):

```
void Residu
(
    Word16 _Y a[], /* Prediction
                   coeffs (Y memory) */
    Word16 x[], /* Speech signal (X
                 memory) */
    Word16 y[], /* Residual signal
                 (X memory) */
    int lg /* Size of filtering */
)
```

In this example, `Word16` should be defined as `_fract`, that is:

```
typedef _fract Word16;
```

and the type `Word16` should be used for 16-bit fractional data only. Integers should be defined with type `int`. The type `_fract` is an extension to ANSI C that allows Tasking C to differentiate between fractional data and integer data. (In the standard reference code, `Word16` is used for 16-bit integer data as well as 16-bit fractional data.)

If the coefficients to be passed to `Residu()` are not already stored in Y data memory, they will need to be copied from X to Y before the call to `Residu()`. (The 10 or 20 cycles required to do this copying will be more than offset by the savings made by the parallel data loads that become possible.)

Inlining of basic math functions. The working reference C code will generally be based on 16-bit fixed-point fractional arithmetic in order to resemble a real-time DSP implementation. However, since the ANSI C language does not support fixed-point fractional arithmetic, all of the basic fractional operations that the codecs require are simulated using a set of subroutines. For instance, a fractional multiply-accumulate operation may be implemented using the subroutine:

```
Word32 L_mac(Word32 L_var3, Word16
var1, Word16 var2).
```

This function, in turn, calls two other subroutines:

NEW
PC-IN-THE-LOOP
PROTOTYPING
PRODUCTS

Don't just predict your real-time system's performance. Prove it.

With the new PC-in-the-Loop™ products, you can accurately test the



New PC-in-the-Loop products enable real-time rapid prototyping of embedded designs on standard x86 PC hardware in any form factor.

performance of real-time embedded system designs on a low-cost PC long before your final target hardware is available. Simulink code generation products convert block diagrams to real-time C code for rapid prototyping. You can

then go directly to your embedded target with our production-quality code.

Learn how PC-in-the-Loop products enable rapid prototyping right on your own desktop.

To get your Embedded Systems Technical Kit, for pricing information, or to buy online now, visit www.mathworks.com/espc.

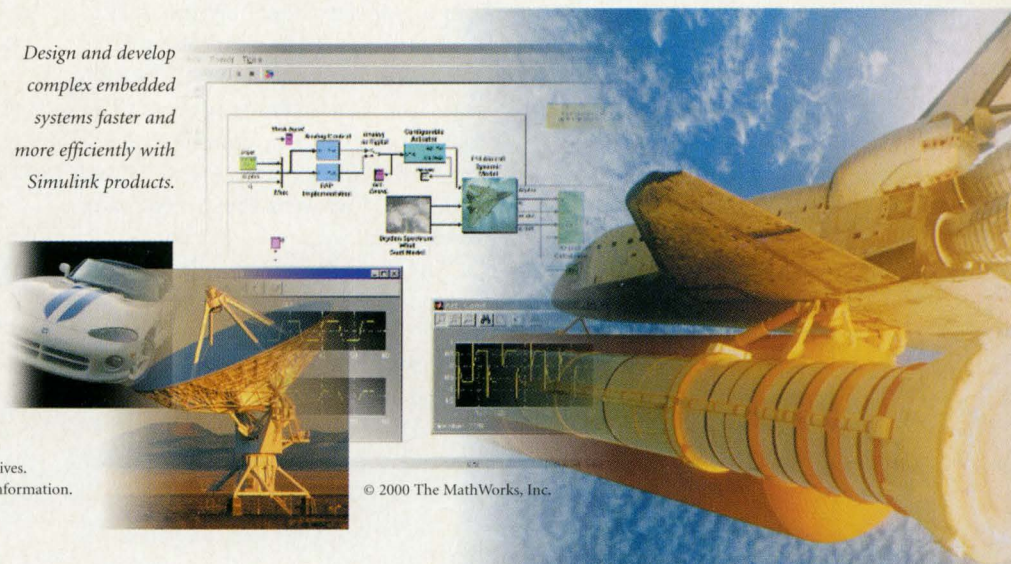
SIMULINK®

*Design and develop
complex embedded
systems faster and
more efficiently with
Simulink products.*



Visit www.mathworks.com/espc
or call 508-647-7000

We have a worldwide network of international representatives.
Visit our Web site at www.mathworks.com/eur for more information.



© 2000 The MathWorks, Inc.


```
Word32 L_mult(Word16 var1, Word16
var2)
Word32 L_add(Word32 L_var1, Word32
L_var2)
```

Therefore, to simulate a fractional multiply-accumulate operation, the reference C code makes three subroutine calls, as illustrated in Figure 2. The simulated fractional multiplication function, `L_mult()`, not only multiplies the variables `var1` and `var2`, but also multiplies the result by two to obtain the fractional multiplication result, and checks for overflow. Similarly, the addition function, `L_add()`, also checks for overflow.

A DSP takes two to three cycles to jump to a subroutine, and two to three cycles to return from that subroutine; therefore, in the previous example, the overhead involved in jumping and returning from subroutines would be 12 to 18 cycles. This number is significant, especially when you consider that each multiply-accumulate instruction can be performed in just one cycle on most DSPs.

This overhead can be removed by one of the following methods:

- Replace each subroutine call by the equivalent C code:

```
L_var3 += var2 * var1;
```

This method is acceptable, but it becomes a long process when all the basic operation calls have to be replaced. A good compiler should generate a multiply-accumulate (MAC) instruction to perform this operation

- Define C macros that will execute the intended operation:

```
#define L_mac(L_var3, var2, var1)
((L_var3) += ((long _fract)
(var2) * (var1)))
```

This approach is similar to the first method but is easier to implement because you need to define the macro only once for each basic

math operation

- Use inline assembly language to perform the more complex arithmetic operations. In some compilers, it's possible to define an assembly language macro that can accept C variable names as parameters, as well as constraints on which type of register the compiler should use for each variable. The exact structure of this type of macro will depend on the DSP and development tools being used. Details will be found in the compiler manual

In each of the previous methods, the DSP must be configured appropriately to produce the correct result. It must be configured to perform saturation on overflow, two's complement rounding, and 16-bit arithmetic. These modes may be set before entering a large block of code, thus eliminating the need for setting and resetting modes for each basic math operation. In addition, the DSP must perform fractional multiplication as opposed to integer multiplication.

The operation of defining C macros in order to replace the subroutine calls must be realized with great care and can be applied to most basic arithmetic operations.

Assembly language functions

Depending on the required performance of the system under development (in terms of both MIPS and memory space), it may be necessary to further optimize the code by writing some of the processing-intensive routines in assembly language. The amount of further optimization that is possible will depend on how well the C compiler deals with the C code, and on how well written the C code is.

The mechanism for calling assembly language functions from C code (and C code functions from assembly language code) will vary from one C compiler to another, and details will be found in the compiler manual. In general, the first few arguments to the function will be passed in certain DSP

registers, and further arguments will be passed on the stack.

As an example, TASKING's C compiler for the DSP56300 family uses a default-calling convention that can be summarized as follows:

- Long arguments are passed in the A, B, X, and Y registers (in that order)
- Integer arguments are passed in the A, B, X0, Y0, X1, and Y1 registers (in that order)
- Pointer arguments are passed in registers R0, R4, R1, R5, R2, ... (in that order)
- The return value will usually be placed in accumulator A

Note that by default the calling function is responsible for saving any registers that must be preserved over the function call; however, it is possible to specify an option which instructs the compiler to switch to a "callee-save" mode, in which the called function must save registers it uses and must restore them prior to returning.

The following code segment shows an example using the function `Residu()` from the GSM EFR codec:

Function prototype:

```
void Residu
(
    Word16 _Y a[],
        /* Prediction coeffs (Y) */
    Word16 x[],
        /* Speech signal (X) */
    Word16 y[],
        /* Residual signal (X) */
    int lg
        /* Size of filtering */
)
```

Function call:

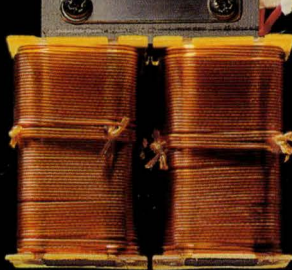
```
Residu (&coeffs[0], &signal[0],
        &residual[0], length);
```

Assembly code for `Residu()`:

`Residu:`

```
; Parameters passed:
; &a[0] Coefficients (Y memory)
; (Passed in r0)
```


Can your Java™
make this ball float?



Small AND fast Jbed can.



And every stroke is a hole-in-one!

Jbed

Pure Java™
based RTOS
for embedded
systems

Jbed runs faster than other Javas because it always compiles, never interprets. Its small memory footprint allows low-cost hardware with less memory. You save time and cost because Jbed offers the productivity of the Java™ language to the embedded world. **Included features are:**

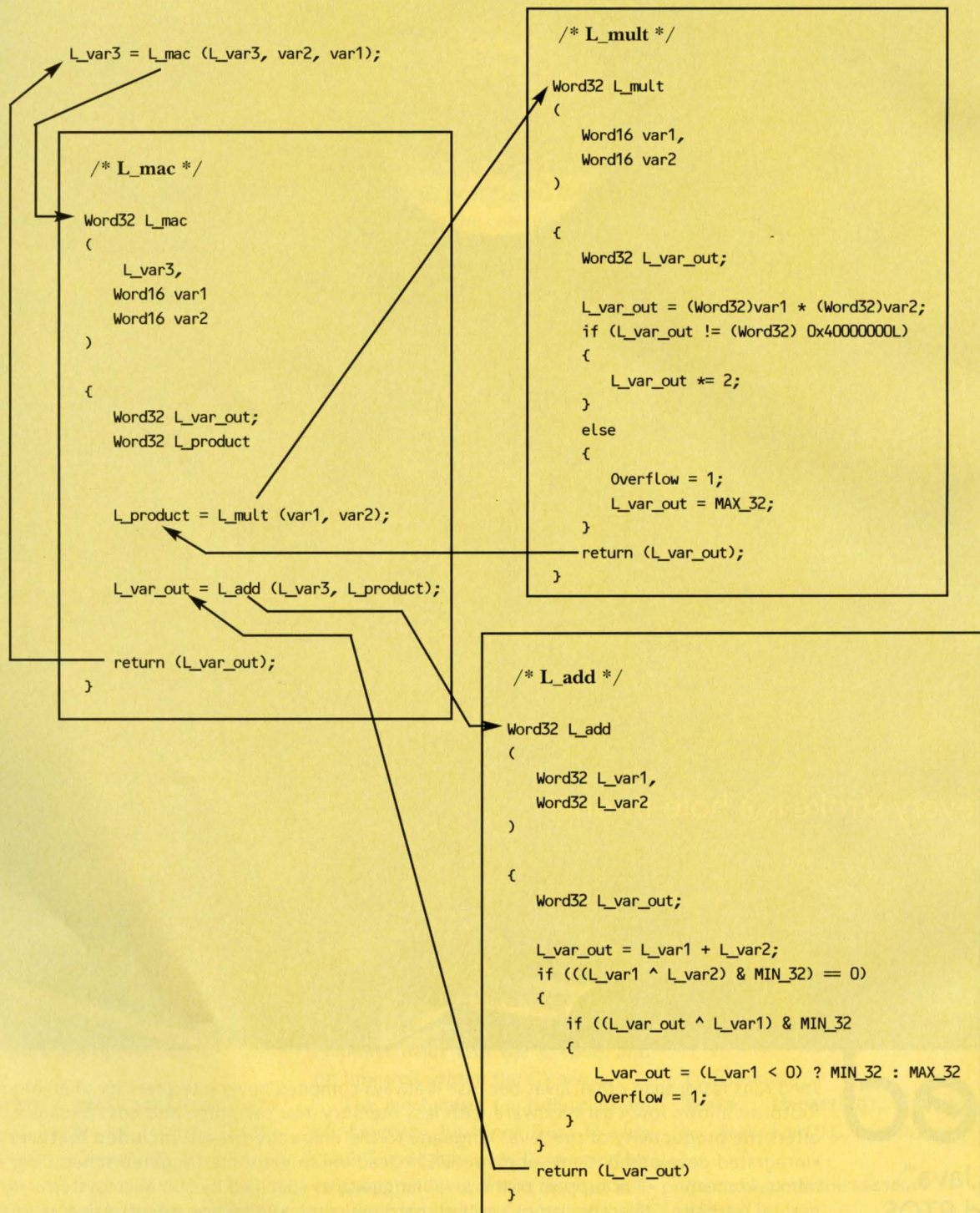
- Integrated development environment (IDE)
- Deadline-driven/time-triggered scheduling with admission testing
- Full support of the Java™ language as specified by Sun Microsystems
- Incremental Garbage Collector compliant with hard real-time
- Reflection, serialization
- Dynamic loading of Java™ byte code
- All standard communication protocols supported
- RTOS and JVM (Java™ Virtual Machine) in one!

esmertec inc., Technoparkstrasse 1, CH-8005 Zürich, Switzerland
www.jbed.com, info@esmertec.com, F +41 1 445 37 30, X +41 1 445 37 34

esmertec

your partner for embedded real-time java

FIGURE 2 Processing involved in a multiply/accumulate operation, as implemented by the GSM EFR ANSI C code. A DSP can perform this operation in one cycle.




```
; &x[0] Input signal (X memory)
    (Passed in r4)
; &y[0] Output signal (X memory)
    (Passed in r1)
; lg Filtering length
    (Passed in a1)
```

```
move #12,n4
    ;Pointer rewind value for r4
move #-11,n0
    ;Pointer rewind value for r0
move x:(r4)-,x0 y:(r0)+,y0
    ;Get first data sample
    and coeff
```

```
do r a1, _end_res;Start outer loop
    mpy x0,y0,b x:(r4)-,x0 y:(r0)+,y0
    ;1st mpy, get next 2 values
```

```
rep #9
    mac x0,y0,b x:(r4)-,x0 y:(r0)+,y0
    ;Sum of products (L_mac)
```

```
mac x0,y0,b x:(r4)+n4,x0
    y:(r0)+n0,y0
    ;Final MAC, dummy reads
    to rewind pointers
```

```
asl #3,b,b ;L_shl(s,3)
rnd b x:(r4)-,x0 y:(r0)+,y0
    ;Round, and get values
    for next iteration
```

```
move b,x:(r1)+
    ;Store result in output array
```

```
_end_res:
rts ;Return (no return value)
```

Note that the address registers used by the C compiler to pass the pointers (r0, r4, and r1) have been used directly in the assembly language code. The only register setup required is the loading of the pointer rewind values into offset registers n0 and n4 to set up address registers r0 and r4 for the next iteration of the outer loop.

As we mentioned previously, the gain to be obtained from writing a function such as `Residu()` in assembly language will depend on how well the C compiler handled the C code version. However, in any case, it is often advantageous to begin with a simple function (such as `Residu()` or

`Syn_filt()`) before embarking on a more challenging section of code, such as the codebook search, in which significant gains are more likely.

Memory map

In a real-time system, placing variables and constants in certain areas of mem-

ory will be necessary. For example, tables of constants will usually be placed in an area of ROM, which may be loaded into RAM during initialization; static variables that need to be preserved from one frame to the next will usually be placed in a certain area of RAM; and temporary variables that

Embed **REAL** PLC Software in your control

REAL PLC functions like

- on-line edit / debug
- Ladder Logic programming, plus Function Block & Structured Text

REAL Embedded Developer Kit:

- source examples
- scalable kernel
- add your own C functions
- use with any RTOS/CPU

REAL PLC experience in
250,000+ installations

Runs on any
PC hardware or
embedded CPU

Visit us at
the Embedded
Systems
Conference
- Chicago -
Feb. 28 - Mar. 2

Don't develop PLC functions on your own.
Get **REAL** PLC programming tools with our IEC 1131 package.
Embed our **REAL** PLC engine in your control.

**Call Now For Your
FREE Evaluation CD**
513-321-9385
www.KW-softwareUSA.com
Cincinnati, Ohio 45208

are used during the processing of one frame but do not require to be preserved for the next frame are usually allocated on a run-time stack. Exactly where each of these areas exists in the DSP memory space will depend on the particular application and what other functions are to be included in addition to speech coding. Normal practice is to write all code (including assembly language code) as relocatable, and to use a linker control file to tell the linker where each section is to be placed in memory. This gives the system developer flexibility to rearrange the memory space without having to modify any code during the integration phase.

Consequently, separating those variables that should be located in ROM from those that should be located in preserved RAM is necessary. One possible configuration is to declare constants (including tables) together in one file, which will be located in a ROM section of memory, and to define state structures for each channel of encoding and decoding that is to be performed, which will be located in a RAM section of memory. These state structures hold all of the information relating to the state of the encoder or decoder, and will include all values that need to be preserved between frames (for example, filter memory, past weighted speech, and so on). Temporary variables that are allocated on the run-time stack are already separate from the other variables, although the placement and size of the run-time stack will also have to be defined.

The exact method for organizing the memory and run-time stack will depend on the development tools being used, which will in turn depend on the DSP being used. However, to illustrate the principle of the state structure, we'll describe a simple example from GSM EFR. First, we'll describe the definition of the state structure types for the encoder and the decoder. Second, state structures will be declared for two channels of

encoding and two channels of decoding. Third, we'll illustrate an example of how a function's prototype might change (to take account of the fact that "static" variables are now stored in the state structure). Finally, we'll give an example of how a state structure for a particular channel would be passed to a function.

1. Types `ENC_STATE` and `DEC_STATE` are defined in a file called `state.h`. This file will be included into any of the ".c" modules that need to access the variables stored in these state structures.

In file `state.h`:

```
typedef struct
{
    /* Speech Vector */
    Word16 old_speech[L_WINDOW -
        L_FRAME];

    /* Weighted Speech */
    Word16 old_wsp[L_FRAME +
        PIT_MAX];

    /* Excitation Vector */
    Word16 old_exc[L_FRAME + PIT_MAX
        + L_INTERPOL];

    /* Filter's Memory */
    Word16 mem_syn[EM];
    etc....
} ENC_STATE; /* Encoder state
              structure type */
typedef struct
{
    /* Excitation Vector */
    Word16 old_exc[L_SUBFR + PIT_MAX
        + L_INTERPOL];

    /* LSPs */
    Word16 lsp_old[EM];

    /* Filter's Memory */
    Word16 mem_syn[EM];

    etc....
} DEC_STATE; /* Decoder state
              structure type */
```

2. In the top level of the encoder and decoder, state structures of type `ENC_STATE` and `DEC_STATE` are declared for each channel to be processed.

In the top-level encoder file (for example, `coder.c`):

```
#include "state.h"

...

ENC_STATE enc_state_1, enc_state_2;
/* 2 channels */
```

and in the top-level decoder file (for example, `decoder.c`):

```
#include "state.h"

...

DEC_STATE dec_state_1, dec_state_2;
/* 2 channels */
```

3. To allow a function such as `Coder_12k2()` (called from the top level of the encoder—in file `encoder.c`) to access the "static" variables contained in the state structure for a particular channel, the function prototype would be changed to allow passing of the address of the state structure:

```
void Coder_12k2
(
    Word16 ana[],
    Word16 synth[],
)

becomes:

void Coder_12k2
(
    Word16 ana[],
    Word16 synth[],
    ENC_STATE *state /* Pointer to
                      state structure */
)
```

4. The call to `Coder_12k2()` (in file `encoder.c`) would then become (for Channel 1):

ASICUS Programmabus.



T rue to its namesake, Spartan™ FPGAs remain synonymous with "a commitment to winning", regardless of the challenge. And armed with that spirit, Xilinx delivers the Spartan-XL family.

The Spartan-XL architecture offers an easy-to-use alternative to traditional ASICs plus the added flexibility and time-to-market advantages inherent in a programmable logic solution.

Ranging in density from 5K to 40K system gates, the Spartan-XL family features on-chip dual port synchronous RAM, while it's 100+MHz performance and flexible I/O structure make it ideally suited for 32 bit 33MHz PCI applications. All this and more for under \$2.50*.

From price to performance, feature set to software, the Spartan-XL device family was born and bred to meet the challenge.

www.xilinx.com



The Programmable Logic Company™

* 250K units:XC505XL-4VQ100C


```
Coder_12k2 (prm1, syn1,
            &enc_state_1);
```

where `prm1` and `syn1` are the input speech and output parameter buffers for Channel 1.

Within the function `Coder_12k2()`, state variables would be accessed in the usual way for structure accesses in C. For example, the last call to

the procedure for modifying files (`coder.c` and `cod_12k2.c` in this case) to make use of state structures. This procedure should be applied to all of `coder.c` and `cod_12k2.c`, as well as to all other files which contain functions that make use of variables (static or global) that need to be preserved for the next frame.

Note that for some of the most

of typical MIPS figures for a selection of speech codecs implemented on single-ALU DSPs in Table 1. These figures are approximate; exact figures will be device- and implementation-dependent.

esp

David Crawford is a senior applications engineer in the Wireless Infrastructure Systems Division of Motorola Semiconductor Products Sector, where he is involved in speech coding applications for mobile systems. He received his PhD from the University of Strathclyde, Glasgow, Scotland, in 1997 for research in adaptive filtering techniques for active noise control. Prior to joining Motorola, David was a research and development engineer in the speech coding group of Enigma, Ltd. (Chepstow, UK). Dr. Crawford is a member of the IEEE and an associate member of the IEE.

TABLE 1 Typical MIPS figures for a selection of speech codecs implemented on single-ALU DSPs. (Actual figures will be device-dependent and implementation-dependent)

Speech codec	Bit Rate	Approximate MIPS	
		Encoder	Decoder
ETSI GSM full rate	13kbits/s	2.5	0.6
ETSI GSM enhanced full rate	12.2 kbits/s	16	2
ITU G.729A	8kbits/s	10	2
ITU G.723.1	5.3/6.3kbits/s	25	3

`Copy()` in the function `Coder_12k2()`—file `cod_12k2.c` in the GSM EFR standard reference code—passes `&old_exc[0]` and `&old_exc[L_FRAME]`. These would therefore need to be passed as follows:

```
Copy (&(state->old_exc[L_FRAME]),
      &(state->old_exc[0]), PIT_MAX +
      L_INTERPOL);
```

The copying is from `old_exc[L_FRAME]` of Channel 1 to `old_exc[0]` of Channel 1, since the state structure passed to `Coder_12k2()` was for Channel 1 (`enc_state_1`). Note that the function prototype for `Copy()` would not need to be changed: the function still receives a source pointer and a destination pointer, and the fact that these are now pointing to locations within a structure is irrelevant.

Note that the address of the state structure can also be passed to other functions from `Coder_12k2()`, as can any variables within the state structure.

The previous example illustrates

recent standards currently emerging, the ANSI C code from the standards body is already written with state structures, thus reducing the burden on the developer.

The bottom line

This article has described some of the techniques involved in implementing speech coding standards in a real-time DSP system. The approach has been general in nature rather than DSP- or development tools-specific; in practice the methodology shouldn't differ to any great extent from what we've described.

The key points can be summarized as the:

- Development of working reference code to run on a PC/workstation
- Allocation of variables to X and Y data memory (or equivalent for a particular DSP)
- Inlining of basic math operations for improved speed
- Assembly language coding of processing-intensive routines and interfacing to C
- Memory map and state structures

Finally, we provide an indication

Emmanuel Roy is a senior DSP engineer in the Wireless Infrastructure Systems Division of Motorola Semiconductor Product Sector. He is involved in developing and benchmarking DSP applications for current and future DSP products. He received his PhD from the University of Strathclyde, Glasgow, Scotland, for research in nonlinear adaptive filtering techniques.

References

1. GSM 06.60 (EN 301 245): "Digital Cellular Telecommunications System; Enhanced Full Rate (EFR) Speech Transcoding."
2. ITU-T Recommendation G.711.
3. M.R. Schroeder and B.S. Atal, "Code-Excited Linear Prediction (CELP): High Quality Speech at Very Low Bit Rates," In Proc. ICASSP'85, pp. 937-940, 1985.
4. ITU-T Recommendation G.729: "Coding of Speech at 8kbit/s Using Conjugate Structure Algebraic-Code-Excited Linear Prediction (CS-ACELP)."
5. ITU-T Recommendation G.729 Annex A: "Reduced Complexity 8kbit/s CS-ACELP Speech Codec."
6. Motorola, Inc. *DSP56300 Family Manual*. Austin, TX, 1999.
7. TASKING C/C++ for DSP563xx, www.tasking.com.

Make Your Embedded Platform Soar!

with

Treck Inc.

Real-Time Internet Protocols for Embedded Systems

We know that schedules are always tight, and that you do not have time to "port" Internet protocols into your embedded environment (let alone write them from scratch). That does not stop marketing people inside your company from asking for Internet functionality in your new product line that is rolling out in the next few weeks. Now you have someone to turn to.

High quality, high performance Internet protocols are what makes us the #1 choice for designers of embedded systems who need Internet connectivity. We design and implement our products specifically for embedded systems. That is why we are three times faster than our fastest competitor. Of course our Internet protocols do not require any porting to your specific platform. This means that you will never need to change a single line of our ANSI "C" code. You do not need an RTOS to use our Internet protocols, but if you have one, we will certainly work with it.

We are very committed to supporting you in your development effort. We know that you do not have time to wait for someone to call days later to get a support answer. That is why we provide the "finest" support in the industry. You can always count on Treck Inc. to help you every step of the way.

Our base product (Treck Real-Time TCP/IP), includes Sockets, TCP, UDP, ICMP, IP, Ethernet, ARP, SLIP and Ping. Treck also provides support for PPP, DHCP, BootP, FTP, and Telnet. All of our Internet products arrive at your door in source code form and are always royalty free.

Call us today to see how Treck Inc. can help you!

Treck Inc.

(800)340-6648 or (513)688-0553

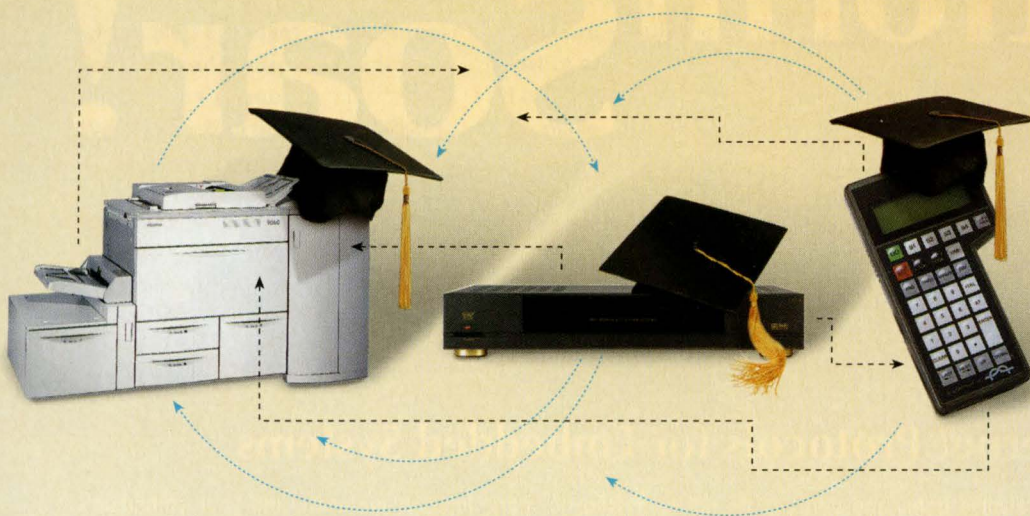
or visit our web page at www.treck.com



SMART NETWORKING SOFTWARE CREATES SMART PRODUCTS

U S Software products
are compatible with:

x86	M*Core
386/486PM	ColdFire
680x0/683xx,	MIPS
DragonBall EZ	C166
PowerPC	CR16
ARM	8096/196
StrongARM	68HC11
SHx	68HC16
SPARCLite	H8/300
I960	8051/80251
NEC V85x	Z80/180



Fast, compact embedded networking solutions

Go beyond standard networking capabilities; U S Software's embedded networking solutions give you complete Internet, Web and network functionality with powerful interface control. Our comprehensive software features small code size and fast performance.

NETPEER™

Distributed Operating Environment

This new and powerful distributed computing software provides real time control of your remote Internet appliance – in less than 16K.

- Remote GUI
- Remote database access
- Notification/diagnostic
- Remote file I/O

USNET®

Embedded TCP/IP Protocol Suite

A fast, compact networking suite, USNET is specifically tuned up for embedded processors and supports a variety of operating systems.

- TCP/IP protocols
- Embedded Web Server
- Internet Access Package
- SNMP

Try NetPeer for yourself!

Download our free NetPeer workstation application at www.ussw.com.



U S SOFTWARE®
EMBEDDED EXCELLENCE

7175 NW Evergreen Pkwy. Suite 100 Hillsboro, OR 97124

TEL: 503-844-6614 • FAX: 503-844-6480 • EMAIL: info@ussw.com • www.ussw.com

Other smart products
from U S Software:

■ SUPERTASK!®

Multitasking RTOS
development suite.
ITRON 2.x/3.x API
available.

■ USFILES®

Windows 95/DOS
compatible file system
with CompactFlash and
CD-ROM support.

■ SOFT-SCOPE®

x86 remote-target
debugger for real
and protected mode
applications.

Call us today at

800-356-7097

and connect with
smarter embedded
networking solutions.

800-356-7097

WWW.USSW.COM

Embedding with XML

The eXtensible Markup Language (XML) is more flexible than HTML, which makes it appropriate in embedded systems that use web technologies.

The end of the twentieth century has marked an age of rapid expansion. With the growth of web-based technological advancements, OEMs, IS/IT professionals, and consumers are saving time and money while increasing their productivity and profitability. Embedded web technology is evolving from a platform for one-to-one element management to one-to-many enterprise device access, control, and configuration via the Internet. The year 2000 will see an expansion of this technology into management of office, mobile, and home appliances. The eXtensible Markup Language (XML) is becoming the lingua franca of this cross-industry expansion.

What is the enterprise?

Embedded systems designers have adopted the web browser as a standard user interface technology for network-attached devices. Web servers are showing up in embedded systems of all kinds. This ubiquity has made possible a revolution in element management, improving the time to market and cost of sophisticated network interfaces for devices. Customers are now beginning to ask that this revolution extend to enterprise management systems. This article explores:

- The qualitative differences between element and enterprise management
- Why the web is suited to enterprise management
- Technologies available to the embedded system designer
- Industry developments that leverage these technologies

Element management: the web browser as network UI

The task of element management is fundamentally to provide the means to monitor and manipulate the state of a device or system. This entails getting access to the data that describes that system, presenting that data in a useful way, and providing any control operations the system requires.

The web has provided the embedded system designer with a solution to a long-standing problem: how to provide a network user interface that is both feature-rich and can be supported across a wide range of user platforms. Prior to the web, a designer had to choose between providing a good interface on a limited set of platforms or limiting the interface to something so simple that it could be widely used (such as telnet or a command line).

The web browser provides a powerful set of user interface tools, expressed in the easily mastered HyperText

The same technologies that are being deployed to integrate advertising, retailing, and customer service on commercial Web sites can be employed to meet the internal needs of the enterprise manager.

Markup Language (HTML). HTML allows the interface designer to display management information in an integrated way, with context and with linkages to related information and help. Web servers designed for embedded systems integrate dynamic information like sensor values into the HTML.

Element management also requires facilities for manipulation and control of the device. In the web, this most naturally takes the form of HTML forms. An HTML form can be mapped to an application interface. Javascript or other active scripting languages can be used to produce control elements such as slider controls not provided by HTML directly, and their value mapped to form elements for transmission to the embedded server.

The benefits of the web as an interface mechanism don't stop with the power of the interface tool, however. The HyperText Transfer Protocol (HTTP), the primary protocol used between web clients and servers, also adds substantial value. It not only provides for exchange of messages, but for an extensible system of message types. This system allows negotiation of the acceptable message types for either participant, description of the freshness of the data in the message, and labeling of whether or not a given message may be safely cached and, if so, for how long.

Enterprise management

Enterprise management is, like element management, monitoring and manipulating the state of a system; but the system is the larger and much more complex interaction of the parts of an entire enterprise. In network management, for example, this means not only the managing of network elements (the routers, switches, and hubs), but also encompassing the state of the network links that connect

them, the topology of those connections, and the policies that express the rules by which each operates.

Enterprise management systems are now also being called upon to provide current information for and interact with:

- Systems that manage capital assets—being able to account for each piece of equipment purchased and which parts of the enterprise use it
- Security management systems—providing integrated management of which users are authorized to use or control each element
- Inventory systems—elements that require consumables like paper and toner for printers being accessible to the systems that optimize inventories

Since no single application can encompass the full range of possible enterprise management functions, a way is needed for data access and exchange among a variety of applications. In addition to the access to primary data sources (the people and devices in the enterprise), enterprise management systems must often interact with and supply data for each other.

Because these enterprise management functions may be implemented as independently developed applications, they will often not share data formats. Each will expect inputs in a particular format, and will produce its own idiosyncratic outputs. The problem of translation between the format produced by each data source and the format required by each possible consumer is one of exponentially increasing size.

The ability of each function to use a common data format therefore increases the utility of all parts of an enterprise management system. Such

an interchange format reduces the complexity of the system from exponential to linear growth.

The web as enterprise management tool

The web is already being integrated into enterprise applications of all kinds. Interfaces to databases and applications allow all kinds of enterprise information to be made readily and remotely accessible. Connections between different sources of information, and connections to the people who provide and use it, are easy to associate on the web. Tools like Cold Fusion and Active Server Pages allow web pages to access databases and even perform sophisticated processing in response to queries.

The same technologies that are being deployed to integrate advertising, retailing, and customer service on commercial Web sites can be employed to meet the internal needs of the enterprise manager.

Devices in the enterprise web

While ready availability of HTML browsers made the web an excellent means of providing for element management, it fell short of providing for enterprise management. HTML is geared toward creating human interfaces, but is not suited to carrying data for exchange between programs. A simple enough display of the status of two ports is easy for an HTML display to render:

```
<table cols="20" width="80%"
  center border>
  <tr><th>Port</th><th>Status
    </th><th>Address</th></tr>
  <tr><td>1</td><td>up</td>
    <td>10.5.6.1</td></tr>
  <tr><td>2</td><td>down</td>
    <td>10.5.7.129</td></tr>
</table>
```

But attempting to import the data in this form into a database or other program would require a sophisticated

LEADERS OF EMBEDDED SYSTEMS TECHNOLOGY

Join the long list of
satisfied EBS customers ...
when you build your next
embedded device!

- All Source Code is Provided

- 100% ANSI "C"

- Royalty Free

- Competitive Prices

Prompt Competent Support

- Consulting

- Satisfaction Guaranteed!

For More Information,

Visit Our Web Site

www.etcbin.com



Are You Prepared For The 21st Century?

Find Out Why The
World's Largest Internet Provider
Chose EBS For Its TCP-IP!!

RTIP - EMBEDDED TCP-IP NETWORK STACK

Internet Protocols	- UDP, TCP, ARP, RARP, BOOTP, ICMP, IGMP, DNS
Device Drivers	- Ethernet, 100 Base-T, Uart, PCMCIA based, PCI based
Kernel Drivers	- AMX [®] , CMX [®] , SMX [®] , Nucleus [®] , Rtkernel [®] , Pharlap [®] , TNT [®] , ETS [®] , TNTRT [®] , PSOS [®] , RTX [®] , RTPX, Polled (no kernel), UCOS [®] , ECOS [®] , VRTX [®] , Threadx [®]
Porting Layer	- Ports easily to any realtime OS/CPU
Dial-Up	- SLIP, CSLIP, PPP, modem support
Application Protocols	- SNMP, DHCP, NFS, FTP, TFTP, Telnet, SMTP, POP3, Virtual & Memory File Systems
Web Server	- Full featured embedded Web Server (with diskless option)
Web Client	- Portable embedded Web Client coming soon!

ERTFS - REALTIME FILE SYSTEM

ERTFS NOW SUPPORTS FAT32

Compatible	- Fully DOS compatible
Fast	- Realtime extensions for demanding applications
Complete	- Full file, directory and partition management API
Devices	- Floppy, IDE, ROMDisk, RAMdisk, PCMCIA Cards
Packages	- FDISK, CHKDISK, MKROM Tool

RTPX - REALTIME PORTABLE EXECUTIVE

Tasks	- Dynamically spawn new tasks
Semaphores	- Signal events with counting semaphores
Mutexes	- Exclusive access to resources
Task Control	- Suspend, stop, resume, reprioritize
Porting Layer	- Ports easily to any CPU

CDFS - CDROM FILE SYSTEM

Compatible	- Read data from ISO9660 CDROM drives
------------	---------------------------------------

1-800-428-9340

EBS/EBSnet, Inc.

P.O. Box 873 • 39 Court Street
Groton, MA 01450

Phone: 978 448 9340 • Fax: 978 448 6376

<http://www.etcbin.com> • email: sales@etcbin.com

Call for our latest brochure with up to date pricing, special features and product demo software!

The goal was to create a language that would make processors and translators easier to write and smaller than those required for SGML or HTML.

parser and knowledge of the HTML presentation conventions being used (for example, that the content of the nearest header in the same column is the label for data in a cell).

XML interchange format

The World Wide Web Consortium (W3C) responded to this need for a data-oriented interchange format by defining the eXtensible Markup Language. XML is a much simplified version of Standard Generalized Markup Language (SGML), an ISO standard and the ancestor of HTML. SGML is widely used for the creation of structured documents. It provides for the definition of document types with rules for the content structure of a document.

Lessons learned in the course of attempting to revise and extend HTML led to the simplifications that make XML documents much easier to parse than either SGML or HTML. The goal was to create a language that would make processors and translators easier to write and smaller than those required for SGML or HTML.

An application designer can easily define an XML representation of virtually any data set, and can exchange the Document Type Definition (DTD) for that representation with anyone who may need to use the data the documents contain.

XML syntax

The syntax of XML documents is simple. A document consists of *content*, which may be anything, and *markup*. The markup consists of tags, which are identifiers surrounded by angle brackets, and may consist of either:

- Matching start and end tags (the end tag has a / after the <):
`<block_element> content`
`</block_element>`

- A tag that has no content (denoted by the / before the >):
`<non_block_element/>`

The opening tag of a block or a tag with no content may contain attributes that are just name/value pairs expressed as an identifier followed by a quoted value (using either single or double quotes):

```
<element attribute="attribute value">  
<element attribute='attribute value'>
```

Elements may contain:

- Non-element content:
`<item>content</item>`
- Nested elements:
`<out> <in1>1</in1> <bar id="x"/>`
`</out>`
- Or uninterpreted data. The special markers `<CDATA[` and `]]>` can delimit data within which the parser does not look for any other markup:
`<opaque>`
`<CDATA[there <can> not be`
`<elements> here]]>`
`</opaque>`

Start and end tags are not allowed to create overlapping content. If a start tag appears within the content of another tag, its corresponding end tag must appear before the end of the first start tag:

```
<outer> <inner> content
```

```
</inner> </outer>
```

but *not*:

```
<first> <second> content
```

```
</first> </second>
```

A simple representation of the previous port status table example might look like this in XML:

```
<ports>  
  <port num="1">  
    <status>up</status>  
    <address>10.5.6.1</address>  
  </port>  
  <port num="2">  
    <status>down</status>  
    <address>10.5.7.129</address>  
  </port>  
</ports>
```

A number of repositories for DTDs have already appeared for the publication of data definitions, and various industry consortia have established exchange mechanisms.

XML offers simple mechanisms for a document to include an identifier for the definition of its markup. The W3C recently added a standard called Namespaces in XML, which allows a single document to reference more than one definition for its markup, disambiguating which components are defined in each by tag name prefixes. This allows vendors to publish data according to an industry standard DTD, combined with vendor- or product-specific extensions defined by a DTD they publish separately. Any namespace-aware processor will be able to distinguish the extensions from the standard data and use the data only as it is able to without ambiguity.

Management data in XML

The Distributed Management Task Force (DMTF) has developed an object-oriented schema system for describing and managing all the components of a distributed system. This class hierarchy allows management applications to understand elements of the system at various levels of abstraction. Because implementing all the base abstractions is easy, an application to do some kinds of monitoring and management of the class of the element, even if the application does not have a complete understanding of it. Method calls and parameters are encoded as XML operations, with an encapsulation mechanism providing for transaction semantics.

These XML encoded operations are transported between management applications and the elements to be controlled by a compatible extension of HTTP, so it can share an implementation of the base protocol for the web interface.

This hybridization of web technologies for program-to-program interaction saves code by sharing it with the web interfaces that provide element management through a browser.

XML adoption

Key to the ubiquitous acceptance of the web as an enterprise management tool and XML as the data definition language is their integration in corporate and desktop applications and application development tools. The following list of vendors represents a sample of applications and application development tools that support XML:

Database vendors:

- Oracle
- Sybase
- Ingress

Productivity and workflow applications:

- Microsoft
- IBM
- Hewlett-Packard

Web content:

- Allaire
- Interleaf

A firm foundation

XML offers the first realistic attempt at integrating management data and linking heterogeneous management systems and tools. The standard is gaining widespread acceptance because it offers so many benefits to network equipment suppliers and their customers.

For suppliers, XML will ease element and enterprise management application development. The specification works with a wide range of hardware, software, and applications, and can scale from small to

large networks. Rather than work with proprietary interfaces, applications will rely on common standards. Instead of porting software to a range of APIs, suppliers will be able to write software once and connect it to any compliant element or enterprise management system. Ultimately, suppliers will be able to embed XML in a new generation of smart devices that will be simpler to configure and easier to manage than current devices because they are web-based.

Customers will also be happy. The move to standards will provide them with a simple, physical, and logical interface to management data. The standards are flexible enough to cover all network management requirements: physical management, billing, and even root cause analysis.

Once widely implemented, XML will enable customers to examine end-

to-end network performance. The markup language will become the foundation that enables companies to manage proactively, rather than reactively, as performance information arrives from various devices.

A change is about to occur in network management space, and proprietary network management products will soon be replaced by open, Internet-based systems. XML has already begun to play a pivotal role in this transformation.

esp

Scott Lawrence is the director of engineering at Agranat Systems. He has extensive experience in embedded systems development and web technology. Most recently, he investigated the application of XML for extended enterprise embedded device access and management. He is also Agranat's representative to the W3C and a contributor to the IETF in the area of security. He can be reached at lawrence@agranat.com.

PROductivity Software

C COMPILERS SUPPORTING:

8051 68HC05 68HC08 COP8™ SX
MELPS740/M38000 Z8 PIC16/17Cxx

BYTE CRAFT
Limited

Features:

Optimizing C Compiler
Integrated Development Environment
Built-in Macro Assembler
BCLink Linker
Runs under: Windows 95/98/NT.
Also available for DOS, HP-UX and Sun Solaris.

Byte Craft Limited
421 King Street North
Waterloo, Ontario
Canada N2J 4E4
Tel: 1-519-888-6911
Fax: 1-519-746-6751
Email: info@bytecraft.com

<http://www.bytecraft.com>

COP8 is a trademark of National Semiconductor Corporation



SEE US AT THE
**EMBEDDED SYSTEMS
CONFERENCE SPRING**
Booth #1209



UNTIL NOW, JAVA FOR EMBEDDED DEVICES
HAS BEEN JUST TOO BIG AND UNPREDICTABLE.

INSIGNIA
SOLUTIONS



You want the power of Java technology. But for your embedded application, you may think it's just too hefty, unpredictable and not fast enough. Insignia Solutions provides the Jeode platform, which is fully compliant with Sun's PersonalJava™ and EmbeddedJava™ specifications and tailored for Internet appliances and embedded devices. Jeode lets you optimize size, performance and predictability for your specific embedded application. Featuring our EVM (Embedded Virtual Machine), Jeode is highly configurable and tunable to achieve optimal performance in resource constrained devices. The EVM provides adaptive optimizing dynamic compilation for fast execution and concurrent garbage collection for predictable behavior. With more than a decade of delivering virtual machines, Insignia is already providing Jeode to Fortune 500 makers of PDAs, set-top boxes, smart storage devices, networking equipment and more. So, can Java technology be small enough, fast enough and predictable enough for your Internet appliances and embedded devices? The answer is a resounding yes! With Jeode. Call **1.800.848.7677** or visit **www.insignia.com** for more.

Code ESP



Embedded Internet Tools

IDE package for Java applications

VisualAge Micro Edition allows developers to build applications that connect small embedded devices to enterprise back-end systems over the Internet. The IDE package includes a team-oriented and repository-based IDE designed for creating Java applications; Java-compatible support for large embedded targets, plus three configurable versions of optimized class libraries; the Smart Linker, which can automatically eliminate unneeded classes and methods when using the optimized class libraries; integrated remote debugger and profiler; run-time optimization and RTOS support; a variety of embedded target-specific optimizations; and an optional micro view user interface framework. The VisualAge Micro Edition IDE is available now. OEM pricing is available under special contract bid with entry pricing starting at \$1,000.

IBM

Raleigh, NC
(800) 722-2227
www.ibm.com/software

Software to deliver HTML content

Prism v. 3.0 enables the delivery of HTML-based content to information appliances. Version 3.0 was specifically redesigned to run across multiple systems concurrently. It's integrated with the Inktomi Traffic Server network cache platform and can be customized to fit other web servers and WAP gateways. Data and images are instantly extracted, compressed, and transformed to meet specific device requirements. It's available now.

Spyglass

Naperville, IL
(630) 505-1010
www.spyglass.com

32-bit platform for smartcards

SmartJ is a 32-bit platform for next-generation, multi-application smartcards that combines direct execution of bytecodes with a native RISC instruction set. SmartJ is a proprietary implementation of the open platform developed by the Multi-Application Secure Smart Card project initiated by MEDEA, a European program of pre-competitive collaborative R&D. At the heart of the platform is the ST22 microprocessor, a 32-bit RISC architecture. The processor core is complemented by on-chip ROM, RAM, and up to 128K of EEPROM, as well as a set of standard peripheral circuits and custom plug-in circuits. The SmartJ platform is supported by an integrated development environment, including SmartJ code generation tools, an instruction set simulator, a cycle assurance simulator, and C, C++, Java, and assembler source-level debuggers.

STMicroelectronics

Lexington, MA
(781) 861-2650
www.st.com

Software for VoDSL

GAO VoDSL (Voice over Digital Subscriber Line) software is targeted at manufacturers of Voice over DSL gateways and customer premise equipment. VoDSL enables compression and packetization of multiple voice signals and includes a data

pump which connects to the digital subscriber line access multiplexer. It also enables processing of the voice in various formats required for frame delay, ATM, and IP standards. A typical VoDSL solution consists of the integrated access device or customer premise equipment, the DSLAM, and a remote switch center which separates the voice and data traffic. The VoDSL software is available in a beta version for \$90,000 to \$190,000 (depending on options) plus royalties.

GAO Research & Consulting

Toronto, Ontario, Canada
(416) 292-0038
www.gaoresearch.com

DSP platform

The CPCI/C5400 digital signal processor platform, based on the Texas Instruments TMS320C5420 DSP, provides telecom designers with the levels of processing power needed to deliver carrier-class voice, fax, and data services to their enterprise and service provider customers. The board is designed to run Telogy's Golden Gateway software package. The CPCI/C5400 is a full-length, 6U cPCI board with a hot-swap friendly PCI interface. The board's DSPs, PowerQUICC, and StrongARM processors are split into four processing blocks, with each block comprising one PowerQUICC, five C5420 DSPs, an optional StrongARM processor, up to 64MB of SDRAM, and 10/100 Base-TX Ethernet interface.

Blue Wave Systems

Carrollton, TX
(972) 277-4600
www.bluews.com

Need Drop In Deeply Embedded Solutions Beyond Protocol Stacks?

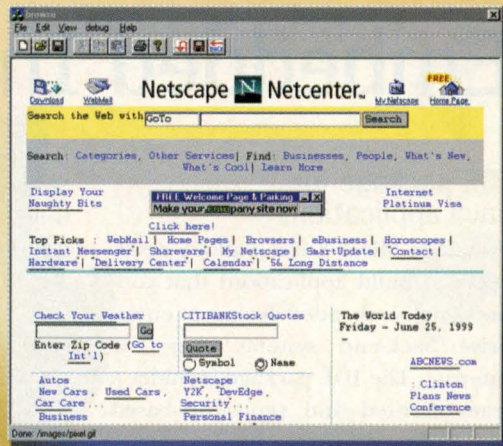


InterNiche's Drop In
Networking Solutions
Can Take You From
Design To Production
Quickly.

Our solutions include all you'll
need for building embedded
Web & DHCP servers,
routing and more.



RTOS & CPU independent
packaged solutions
designed for small footprint
application needs.



Actual NicheView WWW screen shot

Our NicheView™ Deeply Embedded
Browser is the preferred drop in solution
for cost-conscience designs.

NicheView's unmatched patent pending
implementation provides a
fully-operational, fully-featured
Browser in just 30K Bytes.

With support for Graphics and Plug-ins,
your design is not limited by typical
embedded browser technology.

InterNiche Technologies

1340 DeAnza Blvd., Suite 208
San Jose, CA 95129
1.408.257.8014
1.408.257.5692 FAX
sales@iniche.com
www.iniche.com

Embedded Solutions That Drive Change



Dan Saks

Top-Level cv-Qualifiers in Function Parameters

For the past few months, I've been discussing the role of cv-qualifiers (the keywords `const` and `volatile`) appearing in function parameter declarations. I've been focusing on how you can overload functions whose parameter types differ only in their use of cv-qualifiers,¹ and why you might want to do so.^{2,3}

Although cv-qualifiers that appear in a parameter declaration usually become part of the function's signature, they're not always part of the signature. There are times when C++ ignores cv-qualifiers in parameter declarations as it determines a function's signature. In contrast, C never ignores cv-qualifiers in parameter declarations. This subtle difference between C++ and C is my topic for this month.

Passing by value vs. by address

When a C++ compiler encounters a call to an overloaded function, it selects the function to be called by matching the types of the actual arguments in the call against the types of the formal parameters in one of the function declarations. The compiler's overload resolution process often takes cv-qualifiers into account. For example, given three overloaded functions:

```
void f(T *);
void f(T const *);
...
```

```
void f(T volatile *);
```

and this assortment of pointer variables:

```
T *p;
T const *pc;
```

```
g(k);
```

copies argument `k` to parameter `n` without altering `k`. Even if you declared `k` as:

```
int const k = 1024;
```

Although the `const` and `volatile` qualifiers usually add valuable compile-time type information to your programs, they don't always live up to all of your expectations.

```
T volatile *pv;
```

the expression `f(p)` calls `f(T *)`, `f(pc)` calls `f(T const *)`, and `f(pv)` calls `f(T volatile *)`.

The previous example illustrates overloading with cv-qualifiers using functions that pass parameters by address. Let's see how the behavior changes when functions pass parameters by value rather than by address.

For example, a function `g` declared as:

```
int g(int n);
```

has a parameter `n` passed by value. A call to `g` as in:

```
int k;
...
```

the call `g(k)` would work just as well. You can pass a constant argument as a nonconstant parameter, again because passing by value just makes a copy of the argument. In this example, parameter `n` is nonconstant, so `g` can change `n`'s value. However, changing `n`'s value inside `g` has no effect on `k`'s value because `n` doesn't refer back to `k` in any way.

Changing `g`'s declaration to:

```
int g(int const n);
```

has no effect on code that calls `g`. A call such as `g(k)` still copies argument `k` to parameter `n` without altering `k`. It doesn't matter whether `k` is constant. You can always read the value of a constant; you just can't write to it.

Although declaring parameter `n` as

constant doesn't matter to code that calls `g`, it does matter to code within `g`. When `n` is nonconstant, `g` can use `n` as a read-write variable, just as it can use any nonconstant local variable. When `n` is constant, `g` can't alter the value in `n`. However, `g` can still copy `n` to a nonconstant local variable and perform computations in that local variable, as in:

```
int g(int const n)
{
    int v = n;

    // can alter v here

}
```

Declaring a parameter passed by value as constant may affect the function's implementation, but it doesn't affect the function's outward behavior as seen by any caller.

Overloading

The previous discussion raises a number of questions about what it means to declare a pair of functions named `g` as:

```
int g(int n);
int g(int const n);
```

Do you expect `g(3)` to call `g(int)` or `g(int const)`? In other words, should the call choose the `g` that can alter its copy of 3, or the `g` that cannot alter its copy? Or, is it an error to even declare these functions in the same scope? The two `g`'s declared above exhibit identical outward behavior. Therefore, when a C++ compiler encounters a call to `g`, it has no basis for preferring one `g` over the other.

C++ avoids making the choice by treating both `g`'s as the same `g`. Specifically, the compiler ignores the `const` qualifier in:

```
int g(int const n);
```

as it determines the function's signature. Thus, the previous function has the same signature as a function

declared as:

```
int g(int n);
```

Writing both of these declarations in the same scope of a C++ program is not an error. However, defining both of these functions in the same program is an error, which might not be reported until link time.

Here we see a difference between C and C++. In C, declaring both:

```
int g(int n);
int g(int const n);
```

in the same scope is an error. C never ignores cv-qualifiers in a function parameter declaration. In C, these two `g`'s have different function types. The second declaration provokes a compile-time error because C does not permit function overloading.

Top-level cv-qualifiers

In general, C++ does not include cv-qualifiers in a function's signature when they appear at the "top-level" of a parameter type. Here's a bit of background to help you understand what I mean.

Types in C and C++ can have one or more levels of composition. For example, `p` declared as:

```
T *p;
```

has type "pointer to T," which is a type composed of two levels. The first level is "pointer to" and the second level is "T." The declaration:

```
T *f(int);
```

declares `f` as a "function returning pointer to T." This type has three levels. The first is "function returning," the second is "pointer to," and the third is "T."

Different cv-qualifiers can appear at different levels of composition. For example:

```
T *const p;
```

declares `p` with type "constant pointer to T." Here, the `const` qualifier applies only to the first level. In contrast,

```
T volatile *q;
```

declares `q` with type "pointer to volatile T." Here, the `volatile` qualifier applies only to the second level.

In C++, a cv-qualifier that applies to the first level of a type is called a *top-level cv-qualifier*. For example, in:

```
T *const p;
```

the top-level cv-qualifier is `const`, and in:

```
T const *volatile q;
```

the top-level cv-qualifier is `volatile`. On the other hand:

```
T const volatile *q;
```

has no top-level cv-qualifiers. In this case, the cv-qualifiers `const` and `volatile` appear at the second level.

Fundamental types such as `char`, `int`, and `double` have only one level of composition. In a declaration such as:

```
int const n = 10;
```

the top-level cv-qualifier is `const`.

Here's a more precise statement of the way C++ treats cv-qualifiers in parameter types:

The signature of a function includes all cv-qualifiers appearing in that function's parameter types, except for those qualifiers appearing at the top-level of a parameter type.

For example, in:

```
int f(char const *p);
```

the `const` qualifier is not at the top level in the parameter declaration, so it is part of the function's signature.

On the other hand, in:

```
int f(char *const p);
```

the `const` qualifier is at the top level, so it is not part of the function's signature. This function has the same signature as:

```
int f(char *p);
```

In a function declared as:

```
int f(char const *const p);
```

the `const` qualifier to the left of the `*` is not at the top level, so it is part of the function's signature. However, the `const` qualifier to the right of the `*` is at the top level, so it is not part of the function's signature. Thus, the function declared just above has the same signature as:

```
int f(char const *p);
```

It's important to note that C++ does not ignore top-level cv-qualifiers in object and type declarations. For example, in declaring an object such as:

```
port volatile *const p = ... ;
```

the top-level cv-qualifier is `const`. This is not a parameter declaration, so all cv-qualifiers are significant. The object `p` is indeed constant.

More to come

Although C++ ignores top-level cv-qualifiers in parameter declarations when determining function signatures, it does not ignore those cv-qualifiers entirely. I'll explain what I mean by that in my next column. **esp**

Dan Saks is the president of Saks & Associates, a C/C++ training and consulting company. He is also a contributing editor for the C/C++ Users Journal. He served for many years as secretary of the C++ standards committee

and remains an active member. With Thomas Plum, he wrote C++ Programming Guidelines (Plum-Hall). You can write to him at dsaks@wittenberg.edu.

References

1. Saks, Dan, "Using `const` and `volatile` in

Parameter Types," *Embedded Systems Programming*, September 1999, p. 77.

2. Saks, Dan, "Overloading with `const`," *Embedded Systems Programming*, December 1999, p. 81.

3. Saks, Dan, "More on Overloading with `const`," *Embedded Systems Programming*, January 2000, p. 71.

AMD Knows Why CAD-UL is #1.

AMD knows who to count on when they need a partner for x86 embedded development tools. And it looks like they're not alone.

According to VDC, CAD-UL is the number one revenue generator of x86 embedded development tools worldwide.

Why? Because we're committed to providing a seamless development solution from the day you start writing your code to the day you finish testing it. And, oh yeah, CAD-UL tools work!

Do You?

AMD

CAD-UL

- Proven embedded x86 leadership
- Broad portfolio of devices
- Excellent price/performance
- Reference design and code kits

- C/C++ x86 Compiler Systems
- Symbolic Debugging Systems
- IDE: CAD-UL Workbench
- Code Coverage Tools

www.amd.com/embedded
(800) 222-9323

www.cadul.com
(480) 945-8188

BILL GATLIFF

Embedding with GNU: The GNU Compiler and Linker

The GNU compiler, gcc, is capable of producing high-quality code for embedded systems of all types. Here are the gcc features that are most useful for embedded engineers.

A quality compiler, linker, and debugger are critical for the success of any embedded project. In two articles last year, I discussed the benefits of the GNU debugger, gdb.^{1,2} But unlike debuggers, I find it difficult to get excited about cross compilers and linkers. They're excruciatingly technical products. To make matters worse, I tend to stick to the ones that quietly and reliably churn out code, instead of the ones that

don't always get it right but put on a big show in the process.

On the other hand, although the GNU compiler and linker are the strong, silent types, they also sport some nifty features that can really ease the burden of writing solid, portable code for embedded systems. If you stick with me here for a few minutes, I think you will agree.

gcc: the GNU C compiler

In addition to its popularity on the desktop, gcc is perfectly capable of

I find it difficult to get excited about cross compilers and linkers. They're excruciatingly technical products. To make matters worse, I tend to stick to the ones that quietly and reliably churn out code.

producing high-quality code for embedded systems of all types. A complete introduction to all of its features isn't possible in the space I have available here, so I'll only mention the ones that are most useful for embedded systems. See the gcc on-line help files for the rest.

Syntax and potential run-time error checking

The GNU compiler provides several command-line options that control how much trust it will place in your code. The "-Wall" setting is my favorite because it causes gcc to warn you about nearly everything that looks suspicious.

Some other useful options include: "-Wformat" (checks arguments to printf() calls); "-Wcast-align" (warns if a cast can cause alignment issues); and "-Wconversion" (warns if a negative integer constant expression is implicitly converted to an unsigned type).

Inline assembly code

This compiler provides a powerful syntax for embedding assembly language statements in C/C++ source code. In the most basic case, you can insert assembly language instruction(s) into a block of C code as follows:

```
void set_imask_to_6( void )
{
    printf( "switching to interrupt
    mask level 6.\n" );
    __asm__( " andi #0xf8, sr" );
    __asm__( " ori #6, sr" );
    printf( "Interrupt mask level
    is now 6.\n" );
}
```

But that's just the tip of this feature iceberg. The compiler also provides a way to safely refer to C objects in assembly statements, instead of requiring you to predict in advance which

register will be used to store the value of interest.

For example, if you wanted to use the 68881's fsinx instruction in as robust and portable a way as possible, you could do it like this:³

```
__asm__( "fsinx %1,%0" : "=f"
(result) : "f" (angle));
```

The "=f" and "f" are called operand constraints, and they're used to tell gcc both how it must generate the %0 and %1 expressions to make the opcode function properly, and what side effects the operand produces. In this example, they tell gcc that it must use floating-point registers for the values of the variables *angle* and *result*, and that the fsinx instruction returns the answer in variable *result*.

Operand constraints allow mixed assembly and C/C++ statements to work properly even when changes in optimization levels and other compiler settings might cause them to do otherwise. A variety of operands and constraints are available, and several examples are provided in the "Extended Asm" section of the gcc manual.

One other nice thing about gcc's inline assembly language feature is that it doesn't disrupt the compiler's normal optimization processes nearly as much as it seems to with other compilers. To illustrate this, consider the following simple function:

```
int foo( int a )
{
    int b = 10;

    a = 20;
    __asm__( "mov %1, %0" : "=r"
(a) : "r" (b) ); /* a = b */
    return a;
}
```

The assembly language is simply copying *b* to *a*, and so the return value is always 10. With optimizations turned off, gcc emits code that does that explicitly (Hitachi SH-2 code, in this case):

```
_foo:
    mov.l r14,@-r15
    add #-8,r15
    mov r15,r14
    mov.l r4,@r14
    mov #10,r1
    mov.l r1,@(4,r14)
    mov #20,r1
    mov.l r1,@r14
    mov.l @(4,r14),r2
    mov r2,r2
    mov.l r2,@r14
    mov.l @r14,r1
    mov r1,r0
    bra L1
    nop
    .align 2
L1: add #8,r14
    mov r14,r15
    mov.l @r15+,r14
    rts
```

Crank up the optimization level (-O3 -fomit-frame-pointer), however, and the code looks very different:

```
_foo:
    mov #10,r1
    mov r1,r0
    rts
```

In the latter case, gcc's optimizer concluded that the only possible return value was 10, which means that it "understood" the assembly code we supplied and used it to its advantage. Pretty spiffy behavior, and not something I've come to expect from the commercial compilers I've used.

Why didn't gcc just put the 10 into r0 in the first place? Because gcc will optimize around user-supplied assembly code, but it won't omit it. In other words, the `mov r1, r0` is there because of our inline assembly statement, not because gcc put it there.

Controlling names used in assembler code

Occasionally the need arises to have C access to an assembly language object, but the object of interest isn't named in a C-friendly fashion. The following code allows C statements to use a symbol named `foo_in_C` to refer to the pathogenically named assembly language symbol `foo_data` (which lacks a

leading underscore, and therefore can't normally be accessed in C):

```
extern int foo_in_C asm ("foo_data");
```

A similar syntax is used for declarations as well:

```
int bar asm ("bar_none");
extern int foo( void ) asm
("assembly_foo");
```

The first statement causes gcc to create a C symbol called `bar`, but emit assembly code that calls it `bar_none` instead of the usual `_bar`. The second statement causes gcc to use the name `assembly_foo` (instead of `_foo`) when-

ever the C function `foo()` is either called or defined.

Section specification

Many commercial cross compilers allow you to specify the target memory section for declarations by using a command-line option. For example, to place all of a module's constant global declarations into a section called `myconsts`, you often use a command similar to the following:

```
compiler -C"myconst" main.c
```

In my opinion, the problem with this approach is that it invisibly changes the effect of the `const` keyword, which leaves open the possibility that future `const` additions to the module will accidentally end up in the wrong memory space. Furthermore, it forces you to split `myconst` and generic constant declarations into separate modules, which creates a real maintenance headache as a project matures.

In contrast, gcc's approach is to specify allocation sections on a per-declaration basis, using its section attribute language extension:

```
const int put_this_in_rom
__attribute__((section("myconst"
)));
const int put_this_in_flash
__attribute__((section("myflash"
)));
```

In contrast to the command-line approach, this technique allows you to put all of the declarations related to a piece of functionality into the same source module, regardless of their destinations in the target's memory map.

Section attributes can also be used to construct data tables. For example, eCos (the open-source Embedded Cygnus Operating System), uses section attributes to place all device information structures (one of which is allocated in each device driver's source module) into a section called `devtab`, which the operating system then analyzes at run time. Because of

Installing the GNU compiler and linker

The GNU compiler is the only application that can reliably build a cross-platform version of itself, so you'll need to do all of the following on a machine that already has a native gcc installed. Get Linux, or use Cygwin (www.cygwin.com) if you need to build and/or run under Win32.

To build and install the GNU compiler and linker, you first must get the source code. The files you need are:

- gcc: [ftp://ftp.gnu.org/gnu/gcc/gcc-2.95.1.tar.gz](http://ftp.gnu.org/gnu/gcc/gcc-2.95.1.tar.gz)
- linker, assembler, and other utilities: [ftp://ftp.gnu.org/gnu/binutils/binutils-2.9.1.tar.gz](http://ftp.gnu.org/gnu/binutils/binutils-2.9.1.tar.gz)
- run-time library: [ftp://sourceware.cygwin.com/pub/newlib/newlib-1.8.1.tar.gz](http://sourceware.cygwin.com/pub/newlib/newlib-1.8.1.tar.gz)

Untar all the files, login as root, and abide by the following script. If you don't have root privileges on your machine, then add `-prefix=<dir>` to the configure statements, replacing `<dir>` with a pathname that you have permission to write to.

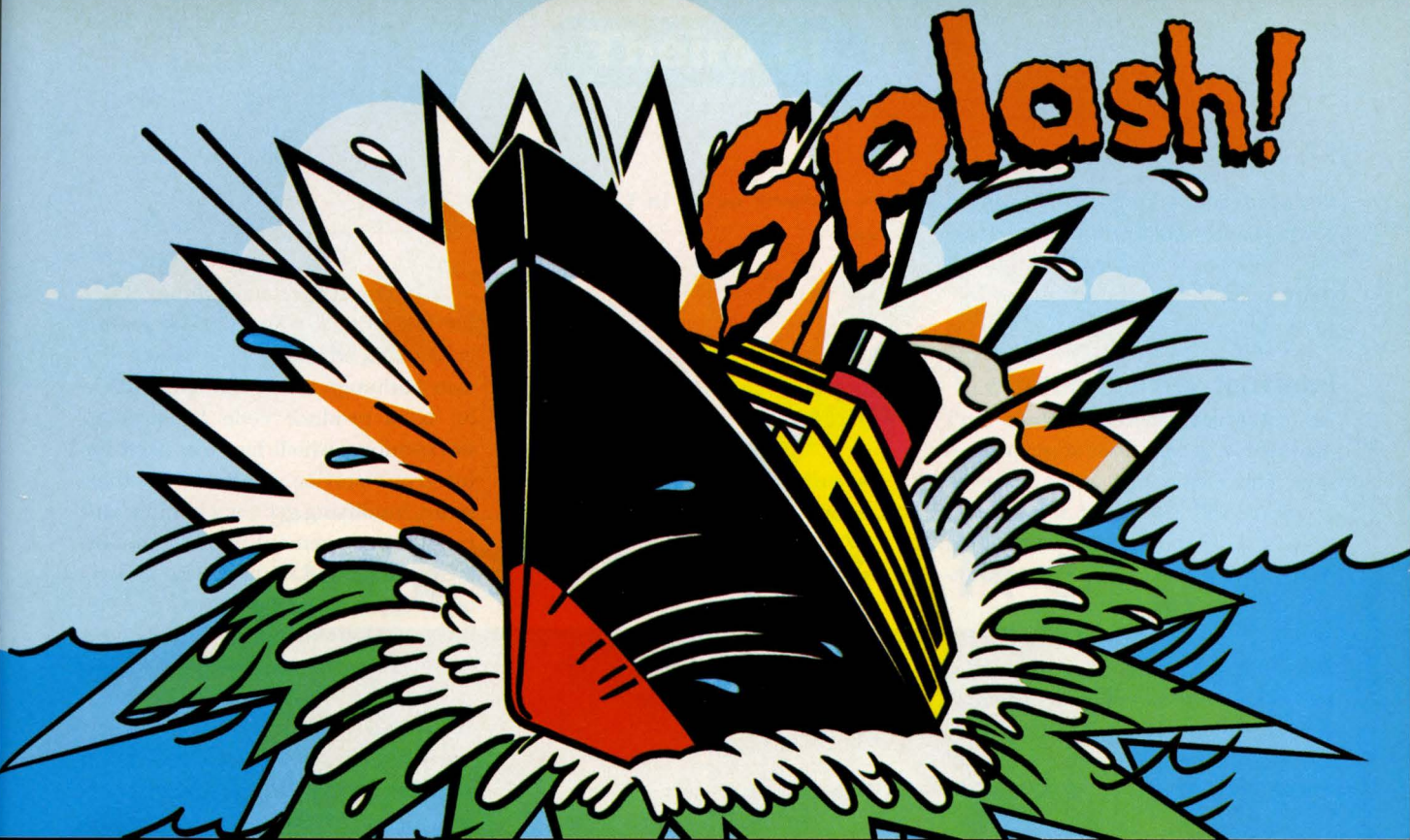
Build the assembler and linker first:

```
cd binutils-2.9.1
configure target=your_target_name
make all install
```

Example target names are `sh-hitachi-hms` for the SH, `m68k-unknown-coff` for CPU32, and so on. See `configure.sub` for information on supported target/host combinations, but beware: the list is long!

Now build the compiler and run-time libraries:

```
cd gcc-2.95.1
rm -rf libf2c
ln -s newlib ../newlib-1.8.1/newlib
configure --target=your_target_name --with-newlib
--with-headers=<ABSOLUTE_PATH_TO_NEWLIB>/newlib/libc/include5
make cross LANGUAGES="c c++" install
```

Next time, Think Lynx.

*Sailing without a reliable RTOS can sink any project.
So next time, you'd better think Lynx.*

That's because Lynx's solid software foundation has proven success in high-availability telecommunications and safety-critical avionics applications. An advanced software process model to speed detection and recovery from faults before they spell trouble. MMU protection that isolates memory space to keep projects sailing smoothly. And the industry's first RTOS to support a commercial off-the-shelf CompactPCI hot swap solution. With scalability down to as small as 33KB Lynx can keep your whole system hierarchy afloat.

In short, Lynx keeps your project sailing full steam ahead. And we're ready to show you how.

For more information contact us: **Email:** info_esp@lynx.com **Phone:** 1.800.255.LYNX Ext. S21.

Web site: www.lynx.com/rtos



Keeping the world running.

© 1999 Lynx Real-Time Systems, Inc.



this approach, adding or removing a driver is as simple as adding or removing a module from the application—there is no “master device driver list” to modify.

Interrupt service routines

An `interrupt_handler` attribute isn't provided by gcc for most target processors. This means that you can't write an entire ISR (interrupt service routine) in C, because gcc won't return from the function using a return-from-exception opcode.

This limitation is not as significant as it sounds. A common workaround is to write a small snippet of assembly language code that saves registers, calls the C code, and then exits with an RTE, like this:

```
void isr_C( void )
{
```

```
    /* whatever we do in C */
    ...
}

__asm__(
    .global _isr
    _isr:

    /* push scratch registers—C
     * preserves the rest
     */
    push r0
    push r1
    ...

    /* call the workhorse */
    jsr _isr_C

    /* clean up, return */
    ...
    pop r1
    pop r0
```

```
    rte
    );
```

Why doesn't gcc make things easier for ISR writers, you ask? The reason appears to be that gcc's primary mission (although this is changing quickly) is to produce code for desktop workstations, which have no need for interrupt service routines. Furthermore, modifying gcc's stack frame and register management implementation is nontrivial, and so most serious GNU users seem to prefer to write a few lines of assembly language here and there, rather than risk additional bugs.

Reserving registers

Sometimes it's nice if you can tell a compiler never to use a particular register, perhaps because you have assembly language functions that require a register value be preserved across C function calls.

As you might have guessed, gcc can do this. Simply put a `-ffixed-REG` on the command line, that is:

```
gcc -ffixed-a7 myprogram.c
```

This capability is also useful if your RTOS reserves a register or two for its own use, or your application has assembly code that needs a high-speed global variable.

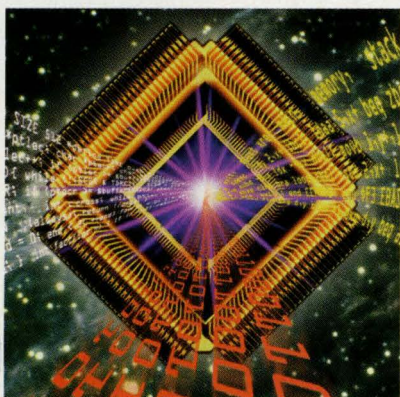
Function names as strings

The standard C preprocessor macros provide minimal information useful for display output at run time. For example, you can describe the `__DATE__` and `__TIME__` of a `__FILE__`, but not much else.

Not only does gcc support these terms, it also adds two of its own: `__FUNCTION__` and `__PRETTY_FUNCTION__`. The two produce the same output in C, but the latter provides more information when it appears in a C++ module. In either case, the information they provide can be useful, especially for diagnostic outputs caused by failed assertions.

For example, this statement:

You always believed there were more intelligent embedded tools out there.



You were right.



E-mail: c-tools@cosmic-us.com
www.cosmic-software.com

Phone: US 781 932-2556
 France 33 1 4399 5390
 UK 44 01256 843400
 Germany ... 49 0711 4204062
 Sweden ... 46 31704 3920

People doubted their existence – yet you continued to search – and now you've found them.

COSMIC C compilers are fast, efficient, reliable, and produce the tightest object code available. Cosmic Software's embedded development tools offer portability for a complete line of microcontrollers. All toolkits include IDEA, our intuitive IDE that provides everything you need in a single, seamless Windows framework.

Add ZAP, our non-intrusive source-level debuggers and minimize your test cycle too. Want proof of their existence?

Download a free evaluation copy of our development tools at www.cosmic-software.com or call Cosmic today.

Cosmic supports the Motorola family of microcontrollers:
68HC05, 68HC08, 68HC11, 68HC12, 68HC16, 68300 and STMicroelectronics' ST7 Family.

Theirs :-)

NETWORKING

Ours :-)

Our networking works! That's why so many leading Internet appliance and application vendors use BSDI. We're the operating system and networking technology inside industry-leading routers, firewalls, load balancers, web accelerators and other Internet devices. BSDI's Internet operating system gives you the reliability, high performance and scalability you need when developing Internet products. We've been building great networking technology since the Internet began, and our advanced features like IPv6, IPsec, packet filtering and rate control make us the most complete networking stack in the industry. If you're developing an Internet product, you need networking technologies that work. So give us a call at 1-800-345-2731 or +1 719 535 8400, e-mail us at info@bsd.com, or visit www.bsd.com.



BSDTM
BERKELEY SOFTWARE DESIGN, INC.

Berkeley Software Design, Inc. • 5575 Tech Center Drive
Suite 110 • Colorado Springs, CO 80919, USA

CALL US FOR A FREE
EVALUATION CD.
1-800-800-4BSD

`printf ("The function %s in file
%s, was compiled on: %s.\n",
__PRETTY_FUNCTION__, __FILE__,
__DATE__);`

yields either:

The function foo, in file foo.c,
was compiled on: Feb 10 1999.

or:

The function int c::foo, in file
foo.cpp, was compiled on: Feb 10
1999.

Debugging information

You always have to tell gcc to include
debugging information in output files,
using the `-g` flag:

`gcc -g main.c`

Without the `-g`, the application will
run but you won't be able to debug it.

ld: the GNU linker

The GNU linker is a powerful applica-
tion as well, but in many cases there is
no need to invoke `ld` directly—gcc
invokes it automatically unless you use
the `-c` (compile only) option.

Like many commercial linkers,
most of `ld`'s functionality is controlled
using linker command files, which are
text files that describe things like the
final output file's memory organiza-
tion. Listing 1 contains an example
linker command script that I will dis-
cuss in detail over the next several
paragraphs. In summary, this script
defines four memory regions called
`vect`, `rom`, `ram`, and `cache`, and the fol-
lowing output sections: `vect`, `text`,
`bss`, `init`, and `stack`.

As with gcc, I don't have the space
to discuss every `ld` feature or support-
ed command, but they're all described
in `ld`'s on-line documentation. Just
type `info ld` after installation.

The linker will use a default com-
mand file unless you tell it to do oth-
erwise. To instruct `ld` to use your com-
mand file instead of its own, give gcc a
`-WL,T<filename>` command during
compilation.

OUTPUT_FORMAT command

This command controls the format of
the output file. A variety of formats are
supported, including S-records (`srec`),
binary (`binary`), Intel Hex (`ihex`), and
several debug-aware formats, like
COFF (`coff-sh` for SH-2 targets, `coff-`
`m68k` for CPU32, and so on).

The GNU linker derives its output
file formatting capabilities from a

development tools

Motorola **HC05, HC08, HC11,
HC12, HC16, m-core
68xxx / 3xx**

Philips **8051XA**
ST Microelectronics **ST7**

HIWARE
Innovation for your Success

**C/EC++/C++ compilers
Real Time Kernel
I/O simulation & stimulation
simulator & debugger**

HIWARE Inc.

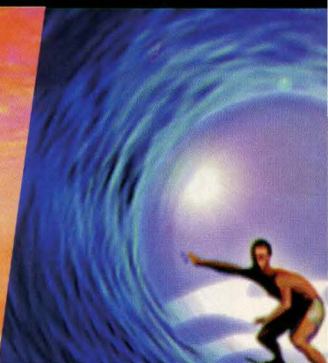
8608 Barasinga Trail
Austin, TX 78749
USA
Phone: (512) - 282 4435
Fax: (512) - 282 4487

info@hiware.com - www.hiware.com

HIWARE AG

Riehenring 175
4058 Basel
Switzerland
Phone: +41 61 690 7500
Fax: +41 61 690 7501

WE CALM THE SEAS OF TECHNOLOGY



SO YOU
CAN RIDE
THE WAVE OF
SUCCESS

WHY PACIFIC SOFTWARES ?

Since 1982, Pacific Softworks has been delivering the quickest and most stable Internet Protocols available in the industry. We're proud of our reputation as the leading supplier of protocol software.

Royalty Free or Royalty based licensing

End-to-End support from the Protocol Specialists

Ported and Tested Compatibility with any Processor,

Kernel or RTOS

Flexible and Scalable solutions to protect your investment

Demonstrated reduction of time to market

Leading the industry with next generation products and solutions World-wide presence

...and, most importantly, we have Happy Customers!

Visit our website for more in-depth information on Fusion, the "off-the-shelf" FastTrack and other leading embedded solutions.

www.pacificsw.com , e-mail: sales@pacificsw.com

Pacific Softworks provides a single supplier solution ensuring that today's successes are the building blocks for tomorrow.

- **Protocols:** TCP/IP (IP v4 and v6*), UDP/IP, DHCP, BOOTP
- **Applications:** FTP, TFTP, TELNET, DNS, SMTP, POP3, RPC, NFS
- **Internet:** PPP, SLIP, CSLIP, CHAP, PAP, MultiLink PPP*
- **Web:** Embedded Web Server, Embedded Web Browser, Web Management
- **Network Management:** SNMP v1/v2, MIB Compiler, SNMP v3*
- **Routing:** RIP/RIP2, OSPF, BGP4, L2TP*, NAT
- **Security:** IPsec*
- **Satellite and Mobile:** Satellite TCP*, and Mobile IP*
- **WAN Protocols:** ISDN, ATM, Frame Relay and X.25 (Integration with Third Party WAN Products)
- **RTOS:** ThreadX™ from Express Logic
- **Development Tools:** Porting Wizard and Installation Wizard

**product in development*

ThreadX™ is a registered trademark of Express Logic, Inc.



Pacific Softworks
our wave... your Success.

California Corporate Headquarters:
tel: 800.541.9508 fax: 805.499.5512

European Headquarters:
tel: 44.1494.432.735 fax: 44.1494.432.728

Japan/Asia-Pacific Headquarters:
tel: 81.3.5669.7722 fax: 81.3.5669.7723

LISTING 1 An example linker command script

```
/* a list of files to link
   (others are supplied on the command line) */
INPUT(libc.a libg.a libgcc.a libc.a libgcc.a)

/* output format
   (can be overridden on command line) */
OUTPUT_FORMAT( coff-sh )

/* output filename
   (can be overridden on command line) */
OUTPUT_FILENAME( main.out )

/* our program's entry point; not useful
   for much except to make sure the S7 record
   is proper, because the reset vector actually
   defines the entrypoint in most embedded systems */
ENTRY(_start)

/* List of our memory sections */
MEMORY
{
    vect : o = 0, l = 1k
    rom  : o = 0x400, l = 127k
    ram  : o = 0x400000, l = 128k
    cache : o = 0xfffff000, l = 4k
}

/* how we're organizing memory sections
   defined in each module */
SECTIONS
{
    /* the interrupt vector table */
    .vect :
    {
        __vect_start = .;
        *(.vect);
        __vect_end = .;
    } > vect

    /* code and constants */
    .text :
    {
        __text_start = .;
        *(.text)
        *(.strings)
        __text_end = .;
    } > rom

    /* uninitialized data */
    .bss :
```

library known as BFD. Use `objdump -i` to find out which formats are supported by your target's version of the linker.

MEMORY command

The **MEMORY** command describes the target system's memory map. These memory spaces are then used as targets for statements in the **SECTIONS** command.

The typical syntax is simple:

```
MEMORY {
    name : o = origin, l = length
    name : o = origin, l = length
    ...
}
```

A one-to-one relationship usually exists between statements in the **MEMORY** command and the number of uniform, contiguous memory regions supported by the target hardware. A typical exception, however, is the processor's reset vector—and in some cases, the entire interrupt vector table—which is usually declared as an independent section so that its final location can be strictly controlled.

SECTIONS command

Statements in a **SECTIONS** command describe the placement of each named output section and specify which input sections go into them. You are only allowed one **SECTIONS** statement per command file, but it can have as many statements in it as necessary.

In the example, the statement:

```
/* code and constants */
.text :
```

starts the definition for a section named `.text`. The statements inside the subsequent curly braces instruct the linker to:

- Create a symbol called `__text_start` and place it at the beginning of the section
- Merge all `.text` and `.strings` sec-

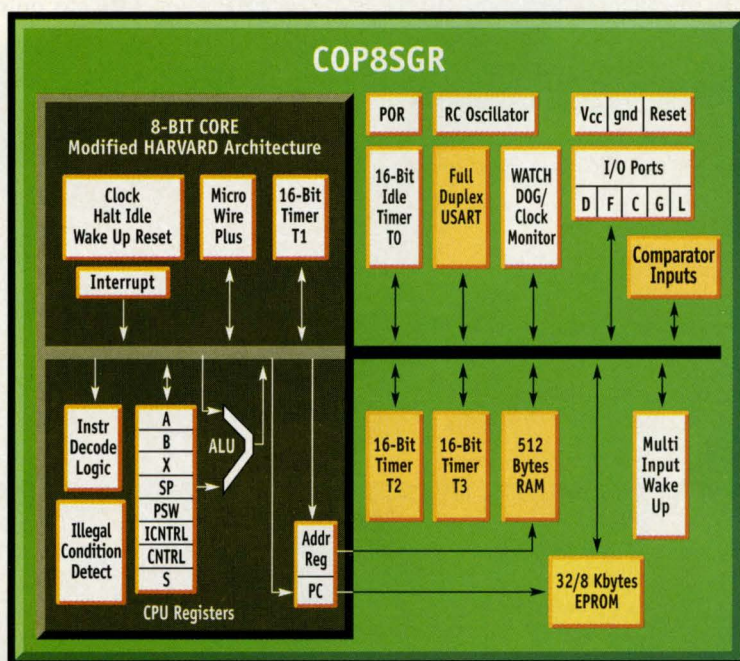
National's COP8.COM


Enables Internet Ready Applications

COP8SGR Plus COP8.COM Connects Your Application Any Place, Any Time

- Internet Connectivity Using emWare® EMIT Technology
- COP8SGR 32K OTP w/512 bytes of RAM
- Full Duplex USART
- 2 Analog Comparators
- User Selectable Watchdog, Oscillator, and Reset Options

COP8 Microcontrollers are Supported by a Full Suite of Development and Programming Tools, Including the IAR® Embedded Workbench Toolset With C-Compiler, Metalink® Emulation and Debugging Tools



©1999 National Semiconductor Corporation. National Semiconductor and  are registered trademarks of National Semiconductor Corporation. emWare is a registered trademark of emWare, Inc. All other trademarks and registered trademarks are the property of their respective owners. All rights reserved.

**COP8.COM is Your Key to
Internet-Controllable 8-bit
Microcontroller Designs**

**For More Information on Ordering
COP8.COM, Other COP8 Products, and
Development Tools**

Visit Our Website at:

**www.national.com/cop8
1-800-272-9959**

tions from the input files into this section

- Create a symbol called `__text_end` and place it at the end of the section

Finally, the statement:

```
} > rom
```

tells the linker to locate the entire section in the memory space called `rom` which, according to the `MEMORY` command, begins at address `0x400`.⁴

The list of input sections can also be file specific. For example, if you added a line like:

```
foo.o (.specialsection)
```

to the `.text` section definition, the linker would also merge into `.text` the section named `.specialsection` from the file `foo.o`.

AT directive

The `AT` directive tells the linker to load a section's data somewhere other than the address at which it's actually located. This feature is designed specifically for generating ROM images, something that's obviously important for embedded systems.

The best way to understand the `AT` directive is by example. So, consider an application that has only one initialized global variable:

```
int a_global = 102;
```

During compilation, `gcc` will declare an integer object `a_global` with the value 102 in the module's `.data` section. But by supplying an `AT` directive for `.data` sections during linking, we tell the linker to assign `a_global` an address in one location (typically RAM), but place its initial value somewhere else (`__text_end`, usually ROM).

The following code initializes `a_global` (and any other initialized global data in an application). This code uses the symbols `__text_end`, `__data_start`, and `__data_end` to find the initial value, determine its size, and place it at its proper place in RAM:

```
extern const char __text_start,
__text_end;
extern char __data_start,
__data_end;
memcpy( &__data_start, &__text_start,
__data_end - __data_start );
```

Now to put it all together. The following command line tells `gcc` to compile a file, `main.c`, and then link it using the linker command file `main.cmd`:

```
gcc -g -WL,-Tmain.cmd main.c
```

LISTING 1, continued An example linker command script

```
{
    __bss_start = . ;
    *(.bss)
    *(COMMON)
    __bss_end = . ;
} > ram

/* initialized data */
.init : AT (__text_end)
{
    __data_start = .;
    *(.data)
    __data_end = .;
} > ram

/* application stack */
.stack :
{
    __stack_start = .;
    *(.stack)
    __stack_end = .;
} > ram
}
```

1 The SBC One Stop Shop 1

PC Compatible SBCs



LCD Touch Screens



PC/104 Add-Ons



Custom Applications



Custom Display Solutions





SBC Microcontrollers
A/D · D/A · PWM



Microcontroller Add-Ons



Microprocessor Training Systems



EMAC, Inc. has been designing Single Board Computers Since 1984, and offers a comprehensive line of products and services for the embedded systems market.

Turn-Key Solutions!

Web: www.emacinc.com • Phone: 618-529-4525 Fax: 618-457-0110

1984-1998
OVER
14
YEARS
OF SERVICE



Build Better Products Faster

... With the Expert's Choice in Embedded Development Tools.

Target CPU's

PowerPC™
ColdFire®
68K/CPU32
M-CORE™
MIPS®
M32R™
SuperH™

DIAB-SDS highly-optimizing compilers, award-winning run-time analysis tools, and comprehensive debug solutions are widely recognized as the preferred tools for demanding embedded applications.

Whether you build with C, C++, or compiled Java™, RTOS or no RTOS, DIAB-SDS' proven tools and support will take you from start to finish in record time.

Our products include: SingleStep™, D-CC™, D-C++™, FastJ™, RTA Suite, SurroundView™

Visit diabsds.com today and sign up for a free test drive. Faster development is just a few clicks away!

 **diab·sds**
An IBM Company
The Embedded Tools Leader

www.diabsds.com

Issues specific to systems built using GNU tools

Although GNU and other open source tools are available at no cost, they aren't necessarily free for unrestricted use. For example, if you link your application with a library released

under the terms of the GPL (GNU Public License), then, according to the terms of the license, you must make your application available in source form as well. Unfortunately, this is the case for the C run-time library (the code for `printf()`, `mal-`

`loc()`, and so forth), most often used with gcc: `glibc`.

A suitable C run-time alternative that isn't restricted in this fashion is a library called `newlib`, available from the Cygnus Solutions archives. According to the terms of this library's license, you may build proprietary applications without disclosing your source code, so long as you acknowledge in a manner appropriate to your application that you're using Cygnus' `newlib`.

Carefully read and understand the licenses for any open source software (or any other software, for that matter) that you use to create or include in your embedded application. If you can't abide by the terms, you can't use the code.

What have you got to lose?

Don't take my word for it—the beauty of GNU is that you can download and try out the tools yourself at no charge. I encourage you to do so and to think seriously about using GNU tools for your next embedded project. **esp**

Bill Gatliff is an embedded consultant and senior design engineer with Komatsu Mining Systems. He welcomes questions or comments via e-mail at bgat@usa.net.

References

1. Gatliff, Bill, "Embedding with GNU: The GNU Debugger," *Embedded Systems Programming*, September 1999, p. 80.
2. Gatliff, Bill, "Embedding with GNU: The gdb Remote Serial Protocol," *Embedded Systems Programming*, November 1999, p. 108.
3. This example, along with several others, is included in the gcc documentation.
4. The name `.text` is traditional. With gcc, the text section actually includes the application's instruction code, constant declarations, and text strings, subject to section attributes supplied in individual modules.
5. Replace `<ABSOLUTE_PATH_TO_NEWLIB>` with where you untarred the newlib sources. for example: `/home/me/cross-gcc-test/newlib-1.8.1)`

TRACE32[®]

IN-CIRCUIT-EMULATORS & DEBUGGERS

ONE SYSTEM FITS ALL!

Standard User Interface

TRACE32-PowerView

Standard User Interface

TRACE32-ICD
the highly cost effective In-Circuit Debugger

TRACE32-ICE
the sophisticated high end In-Circuit Emulator

TRACE32-FIRE
the Fully Integrated Risc Emulator at highest speeds

FULL SPEED !

The main benefits of FIRE:

- First Universal RISC Emulator
- 100 MHz+High Speed Emulation
- High Speed Dual Port Emulation Memory up to 8 Mbyte
- Bus and Program Flow Trace
- Performance-Analysis, Runtime Statistics and Code Coverage Analysis



Embedded Systems 2000
Nürnberg
Feb 16-18, 2000

ESC Spring 2000
Chicago
Feb 28-Mar 02, 2000

Lauterbach Datentechnik GmbH
Fichtenstraße 27
D- 85649 Hofolding, Germany
Fax: ++49-8104-8943-49
Phone: ++49-8104-8943-0
e-mail: info@lauterbach.com

Lauterbach, Inc.
4, Mount Royal Ave
Marlborough, MA 01752
Fax: 508-303-6813
Phone: 508-303-6812
e-mail: info_us@lauterbach.com

LAUTERBACH

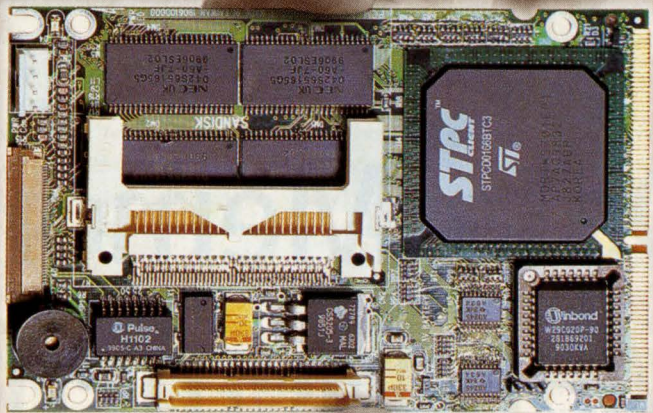


Further information available in the Internet:
<http://www.lauterbach.com>

Compact and Complete CardSize PC

Advantech's new 2.5" size SBC, CPC-2245, is a small-sized, full-fledged and highly-integrated SBC which performs as powerful as a full PC to fit space-critical applications. This ultra-compact SBC offers the benefits of easy integration, installation, maintenance and future upgrade which make it ideal for embedded applications.

2.5" Size SBC



Easy to **Integrate**

Easy to **Maintain**

Easy to **Upgrade**

Easy to **Install**

Full-fledged Board
2.5" Size SBC

CPC-2245

- STPC Client
- 32MB EDO RAM on board.
- One CompactFlash Socket
- 100 Base-T Ethernet
- SVGA Supporting CRT
- 2 Serials, 1 Parallel, 1 IDE, 1 PCI, 1 ISA
- 68mm x 100mm

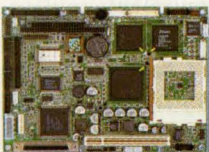
Powered by
Microsoft®
Windows®CE

Socket 370 & 3D

5.25" Size SBC

PCM-9570

- Socket 370 Intel® Celeron™ processors
- 2x AGP 3D & 36-bit LCD
- 100 MHz FSB for Pentium® III processor
- Optional Panel Link & Ultra II SCSI
- 1 IrDA, 2 USBs, 1 FDD, 1 IDE & ATX
- 8" x 5.75"

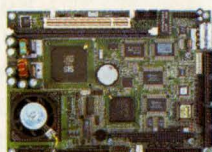


Video Capture

5.25" SBC

PCM-5864

- Pentium® MMX™ processors
- Video-in & TV-out
- On-board LVDS & VGA/36-bit LCD
- AC97 3D Surround Audio
- 100 Base-T Ethernet & FIR
- 8" x 5.75"



486 CPU Card

PC/104 Module

PCM-3346

- STPC Client CPU
- 32 MB EDO SDRAM
- On-board VGA/36-bit LCD,
- 100 Base-T Ethernet
- One CompactFlash Socket
- 3.6" x 3.8"

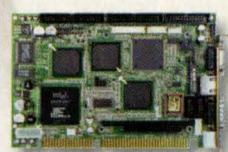


Low Power

Half-size CPU Card

PCA-6751

- Typically 2.08 Amp
- All-Intel embedded solution
- Over 5-year product supply
- Up to 60°C, Fanless
- All-in-one features
- 7.3" x 4.8"



Automation with PCs
ADVANTECH®

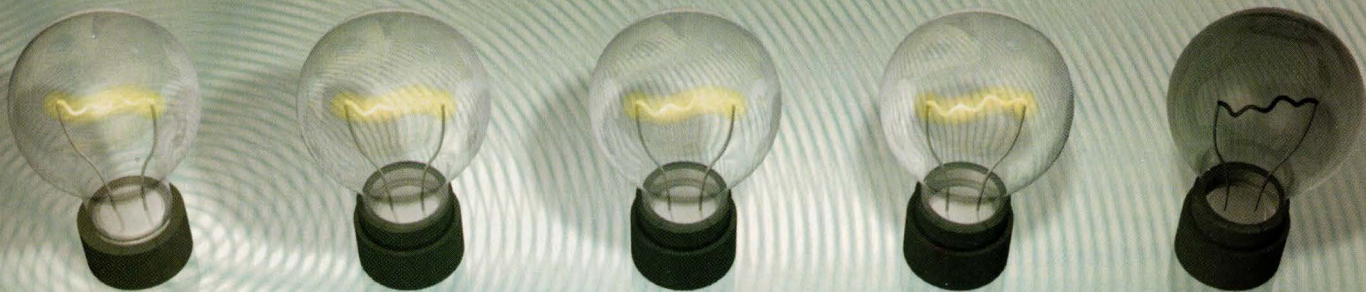
Advantech Technologies, Inc. 1-800-866-6008

Irvine Office: Tel:949-789-7178 Fax:949-789-7179

Sunnyvale Office: Tel: 408-245-6678 Fax: 408-245-5678

EPCinfo@advantech.com

www.advantech.com/epc **Online Shop Open**



Data Memory Paging Management, Part 2

This continuation of last month's article explains a method to detect any potential paging errors in assembly programs.

In Part 1 of this article, I described what paging is, why it exists, and how it can become a problem, and then launched into a method for ensuring that it never gets to be a problem. This article picks up where we left off, detailing the method so that it is useful in larger, nontrivial programs.

Bit addressing

Now I'll add bit addressing to the imaginary processor I described last month because many embedded processors have such instructions, which can even help when paging. We will recognize one more argument type, **bit**, and add two more pseudo-assembly instructions:

```
X<-0    bit    BitName
X<-1    bit    BitName
```

BitName is either of the form **RegName<BitNumber>** or **Page<BitNumber>**. For example, **CounterReg_P2<0>** for the least significant bit of **CounterReg_P2** or **Page<1>** for bit 1 of the current-page register.

The first thing to note is that on most platforms, the bits of a register aren't accessible unless the register itself is. For such platforms, any reference to **bit CounterReg_P2<0>** must have page 2 (and only page 2) active, just as for any reference to **reg CounterReg_P2**.

Also of note is that this new instruction can be a useful way of manipulating **Page**. First of all, if the processor you are using has only two pages, you can replace:

```
A<-X    imm    0
Page<-A
```

with:

```
X<-0    bit    Page<0>
```

and you can replace:

```
A<-X    imm    1
Page<-A
```

with:

```
X<-1    bit    Page<0>
```

Using these equivalent replacements, two program words can be replaced with one. I have completed projects in which 1,022 out of 1,024 words of program memory were used, so this reduction can be important. But even more complex processors, such as this one with four pages, can use bit addressing to shave instructions

Any conditional instructions can be represented by a combination of conditional GOTOs and unconditional instructions.

off a program. Consider the situation at **WhatNext** in Listing 1, which we looked at last month. When the processor reaches **WhatNext**, it may have come from **Label1** (in which case **Page** is 2) or **Label2** (in which case **Page** is 0). The first thing we want to do when we reach **WhatNext** is set **Page** to 2. Given that no other ways to get to **WhatNext** exist, that part of the code can be shortened:

```
WhatNext:                ;#-#-
    X<-1 bit Page<1>      ;--#-
    A<-X reg CounterReg_P2 ;--#-
...

```

The only possibilities for **Page** upon reaching **WhatNext** are 0 and 2. In either case, setting bit 1 of **Page** will leave it with the value 2.

Conditional statements

For most microcontrollers, the only conditional statements are conditional GOTOs; that is, if a certain condition is true (or false) the processor will GOTO a certain place, and otherwise the processor will continue with the next instruction. We shall add one more instruction to the pseudo-assembly instruction set:

```
TEST    imm    AddressLabel
```

If **A<0>** is set, the processor will go to **AddressLabel**. If **A<0>** is clear, the processor will go to the next instruction in program memory. I know it isn't a terribly useful instruction, but it is sufficient to explain how to handle conditional statements. The PIC has a conditional "skip" instruction, which is equivalent to a conditional GOTO to two instructions in front of the conditional instruction. I realize that some processors have conditional instructions that aren't GOTO, but any

conditional instructions can be represented by a combination of conditional GOTOs and unconditional instructions.

The essence of the conditional GOTO is that execution may proceed from two different points in the program. Whatever page (or pages) may be active before the **TEST** may be active at either of these points. Thus, where a **#** appears before a **TEST** instruction, there must be a **#** after the **TEST** and there must also be a **#** after the destination. For example:

```
Test1:                ;--#
    TEST imm Tested1   ;--#
...
Label2:                ;--#
    A<-X reg CounterReg_P2 ;--#
    X<-A reg DataReg     ;--#
Tested1:                ;--##

```

Logical rule 5 (any page that may be active before a flow-control instruction may be active at the destination(s)—for a review of logical and symbolic rules, see Table 1) is applied to the **TEST** instruction. The major difference between GOTO and **TEST** is that **TEST** has two destinations: the label and the next instruction. Symbolically, where a **#** appears in a comment before a conditional instruction, so must a **#** appear in the comment after that instruction and in the comment after the destination label. Symbolic rule 5 is rewritten to include this information:

- S5: when a **#** appears in a comment before a GOTO, a **#** must also appear in the comment after the destination label. When a **#** appears in a comment before a **TEST**, a **#** must also appear in both the comment after the instruction and the comment after the destination label

CALL and RETURN

Add two more instructions to the pseudo-assembly with which we're working:

```
CALL    imm    AddressLabel
RETURN
```

These instructions are implemented exactly as you would expect. A **CALL** instruction will push the address of the next instruction onto a stack, and execution will continue from the label. A **RETURN** instruction will pull an address off the top of the stack and continue executing from there. The behavior of the **RETURN** instruction when the stack is empty may differ from one processor to the next, as may the behavior of

LISTING 1 Example from Part 1

```
#define DataReg rD
#define ChecksumReg_P0 r5
#define CounterReg_P2 r0
#define MaskReg_P2 r3
...
Label1:                ;####
    A<-X    imm    2        ;####
    Page<-A                ;--#
    A<-X    reg    CounterReg_P2 ;--#
    A<-A+X    imm    1        ;--#
    X<-A    reg    CounterReg_P2 ;--#
    GOTO    imm    WhatNext
...
WhatNext:                ;#-#-
    A<-X    imm    2        ;#-#-
    Page<-X                ;--#
    A<-X    reg    MaskReg_P2 ;--#
...
Label2:                ;####
    A<-X    imm    0        ;####
    Page<-A                ;--#
    A<-X    reg    ChecksumReg_P0 ;#-#
    A<-A+X    reg    DataReg     ;#-#
    X<-A    reg    ChecksumReg_P0 ;#-#
    GOTO    imm    WhatNext

```


TABLE 1 Logical and symbolic rules from Part 1

LOGICAL RULES

- L1: variables must be identified and tracked not by ambiguous logical registers, but by unambiguous physical registers
- L2: wherever a paged variable is referenced, the page associated with the variable *must* be active—and equally important, all other pages *must be inactive*
- L3: an instruction that modifies Page will change the active page
- L4: any page that may be active before an instruction that does not modify Page and is not a flow-control instruction may be active after that instruction
- L5: any page that may be active before a flow-control instruction may be active at the destination(s)
- L6: any page that may be active before encountering a label may be active after the label

SYMBOLIC RULES

- S1: any variable name given to a register in page 0 must have the suffix `_P0`, any variable name given to a register in page 1 must have the suffix `_P1`, and so on
- S2: any instruction that references a variable name with a suffix `_P0` must be accompanied with a comment: `;-#—`. Any instruction that references a variable name with a suffix `_P1` must be accompanied with a comment: `;-#—`, and so on
- S3: an instruction that modifies Page has a comment after it that reflects the new value of Page
- S4: where a `#` appears in a comment before an instruction that does not modify Page and is not a flow-control instruction, so must a `#` appear in the comment after it
- S5: where a `#` appears in a comment before a GOTO, so must a `#` appear in the comment after the destination label
- S6: where a `#` appears in a comment before a label, so must a `#` appear in the comment after the label

the CALL instruction when the stack is full. So for simplicity I'll assume that these situations never occur. I shall also assume that the stack is never modified except for the CALL and RETURN instructions themselves.

In any program, every RETURN is associated with a set of CALLs. A RETURN is associated with a CALL if and only if there is a path of execution from the CALL to the RETURN with the following conditions: an equal number of CALLs and RETURNS are along the path, and at no point along the path are there more RETURNS than CALLs. This rule is exactly the association any programmer would make between CALLs and RETURNS, and isn't as complicated as it sounds (at least not in a well structured program). Imagine code like the following:

```
SubA:                ; Point A
...
```

```

                                ; Point B
                                RETURN
...
SubB:                ; Point C
...
                                ; Point D
                                CALL    imm    SubA
                                ; Point E
...
                                ; Point F
                                RETURN
...
SubC:                ; Point G
...
                                ; Point H
                                CALL    imm    SubB
                                ; Point I
...
                                ; Point J
                                RETURN
...
                                ; Point K
                                CALL    imm    SubC
                                ; Point L
```

Consider the path of execution from point K. You can assume that the "... parts of the program can be safely ignored. (They are unconditional pieces of code, they don't have CALLs or RETURNS, they don't have GOTOs or TESTs to anywhere outside of the "... itself, and so on). You will find that the path of execution follows the points in the order K-G-H-C-D-A-B-E-F-I-J-L.

We will focus on the RETURN just after point F (as any programmer would note, the RETURN from SubB). There is a CALL just after point D, and there is a path of execution from this CALL to point F (A-B-E-F). But examining this path of execution shows there is a point at which there are more RETURNS than CALLs (one RETURN, no CALLs), and that is point E. Thus, the CALL at point D isn't associated with this RETURN.

Now consider the CALL just after point K: that path of execution gets to point F (G-H-C-D-A-B-E-F), but along this path there are two more CALLs and only one RETURN. Thus, the CALL just after point F is not associated with this RETURN either.

Finally, consider the CALL just after point H. This path of execution (C-D-A-B-E-F) goes through one more CALL and one RETURN before it gets to point F. Thus, the CALL at point H is associated with the RETURN after point F. Neither of the other CALLs are associated with this RETURN. The RETURN at point F is essentially a RETURN from SubB, and the only CALL to SubB is the one at point H.

Subroutines

As far as paging is concerned, every CALL must be treated as a GOTO. Every RETURN must be considered a GOTO to the instruction after any (and all) associated CALLs (that is, a RETURN may have many destinations, in the same way that a TEST has two destinations). For example, a subroutine might exist to add a data byte to a checksum. Given declarations that I previously discussed, such a subroutine might be written:

Don't gamble with success ...



Hitex deals you winning cards!

USB Agent - USB Analyzer

The USB Agent is a full-featured USB-Protocol analyzing instrument ...

A development tool that captures, analyzes and intelligently displays all USB-signals. Right performance - Right price!

In-Circuit Emulators for ...

MX430 family:

For MSP430 microcontrollers, these emulator systems support all members of this TI family of low-power microprocessors, including the ultra-low-power MSP430x11x derivatives. These instruments are proving to be very popular.

C166 family:

A modular family of powerful in-circuit emulator systems for the Siemens C16x variants featuring the most advanced adaptation technology: PressOn. Our C161 system with a 64 K trace buffer module is available for less than \$3,500. Also supports ST-10 microcontrollers.

68HC12 family:

For 68HC12 processors; supports maximum speed and all operating voltages; plug and play (no jumpers, no switches); additional BDM interface cable; flash programming built-in; PlugOn trace with 32k frames. Move up to super advanced features with DBoxHC12.

8051 family:

True real-time emulation for more than 500 variants from all manufacturers. Including derivatives like Atmel 89S8252, Intel 8x931x and Siemens C505C.

and more:

251, 68k, CPU-32, 80x86, 68HC11, Pentium® Processor

DProbe167 and DBox167

DProbeHC12

MX51/AX51

hitex

DEVELOPMENT TOOLS

Hitex USA
710 Lakeway Dr., Suite 280
Sunnyvale, California 94086

Tel.: (800) 45-HITEX
Tel.: (408) 733-7080
Fax: (408) 733-6320
E-mail: info@hitex.com

Hitex Germany

Tel.: (0721) 9628-133
Fax: (0721) 9628-149

Hitex UK

Tel.: (01203) 69 20 66
Fax: (01203) 69 21 31

Hitex Asia

Tel.: (65) 74 52 551
Fax: (65) 74 54 662

www.hitex.com

"The best emulator I have ever used!"

Follow this URL to a special promotion:

<http://www.hitex.com/hitools/deal.htm>

Benefit from our world-wide support:

Austria Tel: +43-1-259727/20 Belgium Tel: +31-77-3078438 PR China Tel: +86-10-62383376 Czech Republic Tel: +420-2-66316661 Denmark Tel: +45-43-424742 France Tel: +33-1-39611414
India Tel: +91-22-85208171 Israel Tel: +972-3-7657666 Italy Tel: +39-0434-633500 Malaysia Tel: +603-7167591 Netherlands Tel: +31-77-3078438 Pakistan Tel: +92-51-822075 Poland Tel: +48-22-7556983
Russia Tel: +7-812-3252792 Singapore/Indonesia Tel: +65-7496168 South Korea Tel: +82-2-645-0386 Spain/Portugal Tel: +34-91-6401234 Sweden/Norway/Finland Tel: +46-87-404580
Switzerland Tel: +41-1-3086666 Taiwan Tel: +886-2-26983456 Turkey Tel: +90-312-4472700 PBX


```

UpdateChecksum:          ;####
A<-X imm 0               ;####
Page<-A                  ;#---
A<-X reg CheckSumReg_P0 ;#---
A<-A+X reg DataReg       ;#---
X<-A reg CheckSumReg_P0 ;#---
RETURN

```

We write the subroutine so that it doesn't matter what page is active when the subroutine is CALLED. If there are several CALLs to UpdateChecksum, the code could look like this:

```

...
;---#
CALL imm    UpdateChecksum
;#---
...
;---#
CALL imm    UpdateChecksum
;#---
...

```

We can observe that these code fragments all work together. When there is a # in the comment before a CALL, there is a # in the comment after the destination. When there is a # in the comment before a RETURN, there is a # in the comment after the associated CALLs. So let's now add CALL and RETURN to our collection of logical rules:

- L7: a CALL is to be treated as a GOTO, as far as paging is concerned
- L8: a RETURN must be treated as a GOTO with multiple destinations, those destinations being any and all instructions immediately after an associated CALL

These logical rules translate to the symbolic rules:

- S7: when a # appears in the comment before a CALL, a # must

appear in the comment after the destination

- S8: when a # appears in the comment before a RETURN, a # must appear in the comment after any and all associated CALLs

However, an even simpler subroutine shows that the tools described so far aren't always adequate for creating the shortest and best solutions to paging tracking:

```

IncRegB:
A<-X reg rB
A<-A+X imm 1
X<-A reg rB
RETURN

```

The program may have two calls to IncRegB:

```

...
;---#
CALL imm IncRegB
; Point A
...
;---#
CALL imm IncRegB
; Point B
A<-X reg CounterReg_P2
...

```

The best way to complete paging comments for this program would be to start with the CALLs. Assuming that the line before the label IncRegB is GOTO or RETURN (that is, the flow of control to IncRegB never comes from the instruction immediately before), and there are no other references to IncRegB in the program, the subroutine should be written:

```

IncRegB:          ;---#
A<-X reg rB       ;---#
A<-A+X imm 1      ;---#
X<-A reg rB       ;---#
RETURN

```

So far, there is no problem. The register rB is unpaged, so we know the paging is acceptable for this part of the program. Next, we handle the

What do the leading silicon vendors know about BIOS?

The leading silicon vendors know that the right BIOS is key to the success of your embedded design -- and that configurability is key to the right BIOS.

That's why AMD, Intel, and STMicro ship General Software's Embedded BIOS preinstalled on their embedded platform evaluation boards.

With over 400 configuration options, Embedded BIOS offers the advanced configurability you need to run your custom target environment without editing the core BIOS source code.

Contact us today for information and a free BIOS binary sample for your standard reference design.

Embedded BIOS™

ADAPTATION KIT: Full source code automatically configured with over 400 build parameters, using BIOStart™ expert system.

CORE BIOS FEATURES: ROM, RAM, Flash disks, Setup system, console redirection, manufacturing mode, WinCE loader, configurable PCI, integrated debugger, modular callouts to chipset, board, and CPU-level modules.

CHIPSETS: ALI Aladdin V, Finali; AMD 186, SC300, SC400, SC520; INTEL 386EX, 430HX/TX, 440BX; NSC MediaGXm, Geode GXLm; STMicroelectronics STPC family.

IDEAL FOR: DOS, Win95, Win98, WinNT, Embedded NT, WinCE, Linux, Embedix, VxWorks, QNX, pSOS, and all x86-based operating systems.

www.gensw.com

sales@gensw.com

1-800-850-5755

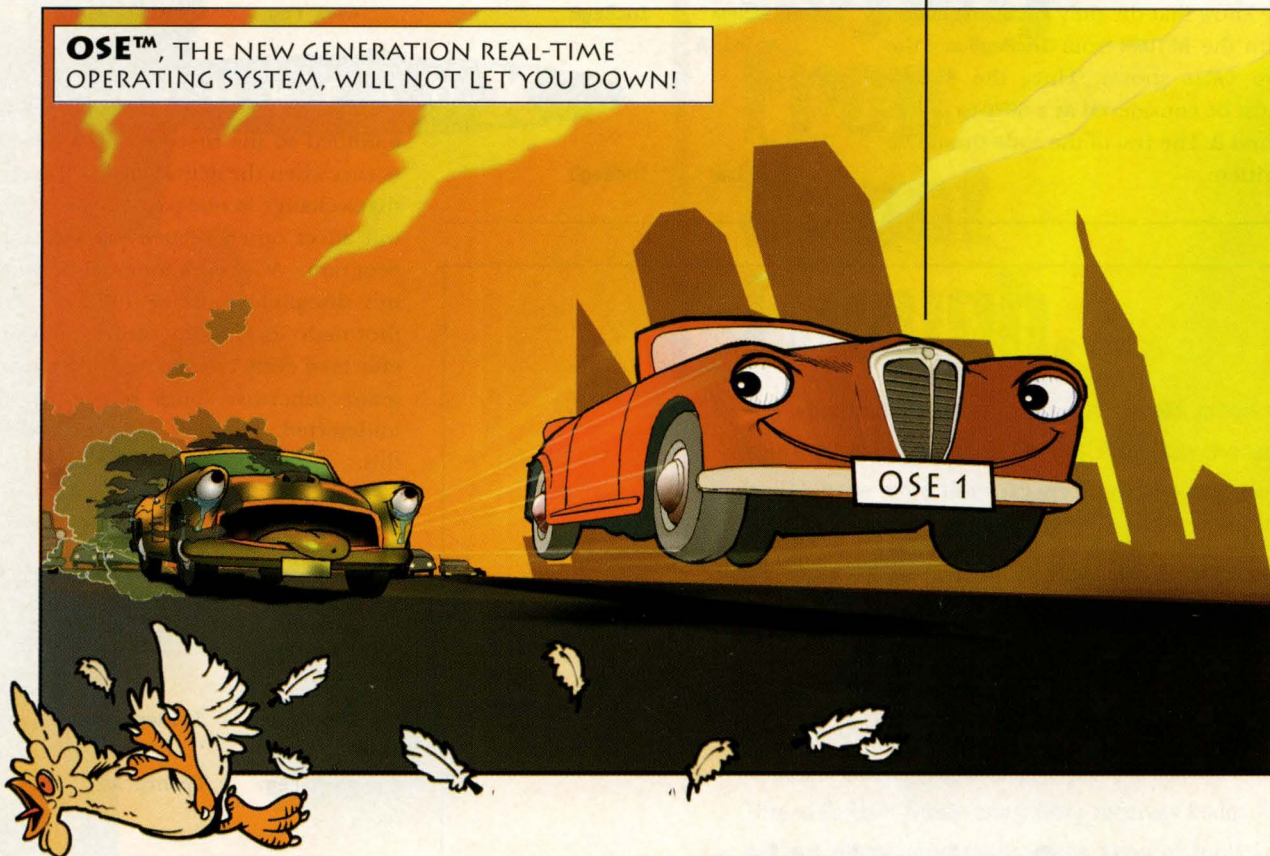
(1-425-454-5755)





– Conventional RTOS? ...sorry!

OSE™, THE NEW GENERATION REAL-TIME OPERATING SYSTEM, WILL NOT LET YOU DOWN!



OSE THE NEW GENERATION RTOS!

FASTER

Streamlined for extreme reliability and speed, OSE's kernel and TCP/IP stacks blow away the competition. From the RTOS to the tools, OSE sets tomorrow's standards for small size and high performance.

TOUGHER

As the only fault-tolerant RTOS, OSE supports mission-critical real-time systems, allowing complete non-stop recovery from hardware and software failures AND hot swaps – critical for high-availability functionality.

LONGER-LASTING

OSE is heterogeneous, scalable, and distributable, protecting your investment by allowing your application to grow from one CPU to hundreds.

MORE POWERFUL

OSE's kernel offers automatic supervision, dynamic reconfiguration and integrated error handling, letting you focus on your core competency: designing applications.

SIMPLER

OSE's powerful ultra-efficient message-based architecture lets you write nearly every bit of application code using only eight system calls.

SAFER

OSE is the world's only RTOS that is safety certified to the demanding specifications of IEC 61508. OSE is also being certified according to the stringent DO 178-B.

PROVEN

Millions of products worldwide are already taking advantage of OSE, including the top brands in telecommunications and process control. OSE is the RTOS of the future.

RETURN. Under the conditions that have been described for this program, we know that the only CALLs associated with the RETURN from IncRegB are the two CALLs shown. Thus, the RETURN must be considered as a GOTO to points A and B. The rest of the code should be written:

```

...
;---#-
CALL imm      IncRegB
; Point A
;---#-
...
;---#-
CALL imm      IncRegB
; Point B
;---#-
A<-X reg      CounterReg_P2
...

```

Imagine now that the program is modified so the first CALL to IncRegB occurs when the active page is 1. Note that a change in one part of a program will affect other, related parts of the program. A programmer following this discipline must be diligent and thorough in tracking down all side effects of every change made to a program; otherwise, bugs will creep in undetected. The new code looks like this:

```

...
;---#-
CALL imm      IncRegB
; Point A
...
;---#-
CALL imm      IncRegB
; Point B
A<-X reg      CounterReg_P2
...

```

The first step again is to consider the CALLs. The subroutine IncRegB should now be written:

```

IncRegB:      ;---##-
A<-X reg rB   ;---##-
A<-A+X imm 1   ;---##-
X<-A reg rB   ;---##-
RETURN

```

Again, register rB is unpaged, so this code will operate correctly. Now consider the RETURN—as before, this RETURN is only associated with the CALLs shown. Therefore, the code should be written:

```

...
;---#-
CALL imm IncRegB
; Point A
;---#-
...
;---#-
CALL imm IncRegB

```



Our inspiration.

**Powerful tools. Great productivity.
Integrated price.**

Paradigm introduces all the tools you'll need for x86 integration in one package.

Paradigm C++ is alone in offering a complete integrated development environment that includes all the tools you need to get your x86 embedded application jump started. Editing, project management, debugging, compiler, assembler, version control and more, all fully integrated into the powerful Paradigm C++ development environment.

If you are tired of wasting time on non-integrated tools, then Paradigm C++ is where you want to be. Download a copy of Paradigm C++ from <http://www.devtools.com/pcpp> and see the future of x86 development tools today.

Paradigm

Paradigm Systems
3301 Country Club Road
Suite 2214
Endwell, NY 13760

1-800-537-5043

Phone 607-748-5966
Fax 607-748-5968
info@devtools.com
<http://www.devtools.com>



THREADX

Being Royalty Free Is not Enough.

You work under a lot of pressure to meet production demands and schedules. So the RTOS you choose has to have more than one outstanding quality to make you feel comfortable.

ThreadX, the leading royalty-free, source-code RTOS gives you more, much more...Like state-of-the-art technology with preemption-threshold and a picokernel design that automatically scales to your needs. Processor support for most processor families. Documentation so clear and concise that ThreadX's User Guide constantly receives kudos from our customers. And such a wide array of tools that chances are there is ThreadX-aware debugging already built-into your favorite tool chain.

Call 888.THREADX or visit www.threadx.com and find out more about ThreadX, the RTOS with all the features to make you comfortable.

Processors Supported

- Win32 **NEW!**
- ARC
- ARM
- StrongARM
- Thumb
- Power PC
- ColdFire/68K
- x86
- Hitachi SH
- MIPS
- NEC V8xx
- TinyJ
- SPARC
- M-Core
- TriCore
- SHARC
- TMS320C6x
- TMS320C54x

ARM

MIPS

Hitachi

PowerPC

ColdFire

the comfortable RTOS



eL

Express Logic, Inc.

tf 888.THREADX • fx 858.613.6646 • www.expresslogic.com

LISTING 2 UpdateChecksum rewritten to demonstrate more complex and useful subroutines

```

UpdateChecksum:                                ;{Page = P'}
    A<-Page                                    ;{Page = P' and A = P'}
    X<-A    reg    rE                          ;{Page = P' and rE = P'}
    A<-X    imm    0                          ;{Page = P' and rE = P'}
    Page<-A                                    ;{Page = 0 and rE = P'}
    A<-X    reg    ChecksumReg_P0            ;{Page = 0 and rE = P'}
    A<-A+X  reg    DataReg                    ;{Page = 0 and rE = P'}
    X<-A    reg    ChecksumReg_P0            ;{Page = 0 and rE = P'}
    A<-X    reg    rE                        ;{Page = 0 and A = P'}
    Page<-A                                    ;{Page = P'}
    RETURN

```

where they aren't necessary. The only purpose of the `A<-X imm 2` and `Page<-A` instructions is to keep the comments in line. "This is unacceptable," I hear you say, "adding instructions to a program that aren't necessary for the correct operation of the program." I quite agree.

The next step is to introduce a little higher-level logic, and a concept called "dynamic scope." The trick I am about to show is a new symbolic notation, so the previously written symbolic rules 7 and 8 will be superseded. However, logical rules 7 and 8 are still adhered to. I shall rewrite the subroutine and code as follows:

```

; Point B
;---
A<-X imm 2    ;---
Page<-A       ;---
A<-X reg CounterReg_P2 ;---
...

```

You probably anticipated this scenario. Clearly, the subroutine doesn't alter the paging, but the comment after a `CALL` is different from the comment before. This setup can lead to extra instructions for setting the page

```

IncRegB:                                ;{Page = P'}
    A<-X    reg    rB                    ;{Page = P'}
    A<-A+X  imm    1                    ;{Page = P'}
    X<-A    reg    rB                    ;{Page = P'}

```

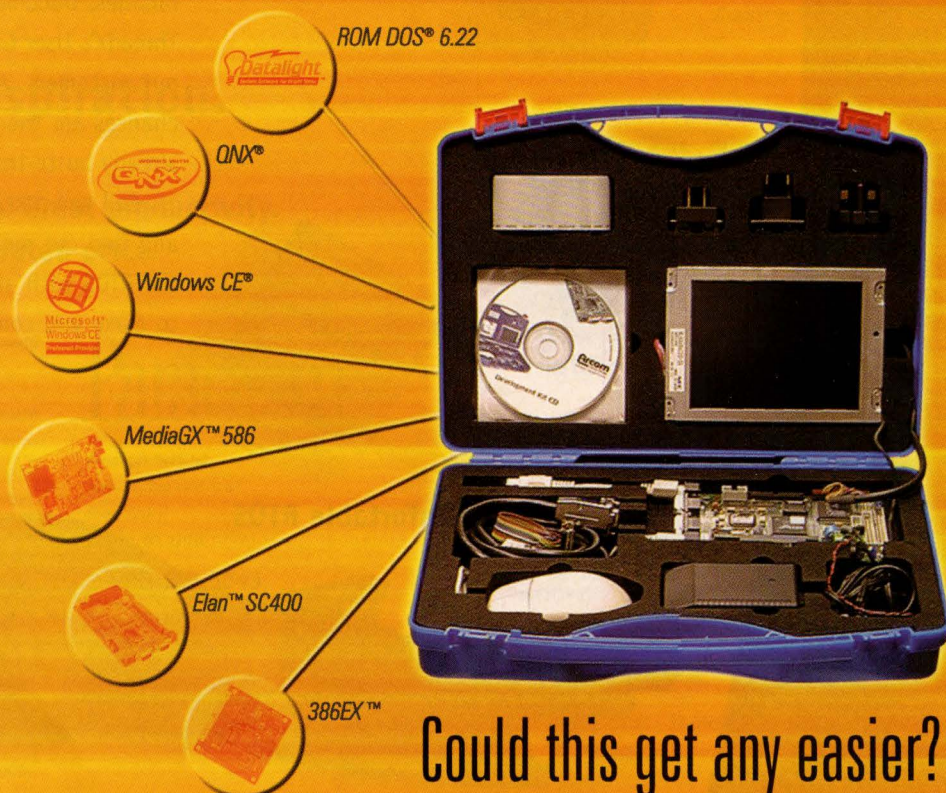
You Choose the
Operating System.

You Choose the Single
Board Computer.

We Do the Rest.

Arcom
CONTROL SYSTEMS

1.888.941.2224 • www.arcomcontrols.com



Could this get any easier?

Hardware and software integration provided by Arcom Controls

RESUME

Name: HI-TECH C
Address: 7830 Ellis Rd., Suite 105, Melbourne, FL 32904
Phone: 800 735 5715
Fax: 407 722 2902
Email: hitech@htsoft.com
Web site: <http://www.htsoft.com/>
DOB: Jan 1st, 1984
Salary: Peanuts. Really. But a small signing bonus would be nice.

* Interview ✓
* References ✓
* Qualifications ✓
* Call Tomorrow

Skills: Highly experienced in the translation of C programs into machine code for a wide variety of microcontrollers. Can produce working code up to ten times faster than a human assembler programmer. Skilled in debugging complex programs.

Education: George Institute of Technology, Cornell University, University of California, State University of NY, University of East London, MIT, Sydney University, University of Waterloo, Worcester Polytechnic, Ecole Polytechnique, Fachhochschule Hannover and many others too numerous to mention here.

Employment: Currently employed at Microchip Inc, Cisco Systems, NASA, Federal Reserve Board, British Railways, Intel, Siemens, Motorola, Philips, Garmin, Hitachi and way too many more to list here. I like to spread myself around.

Languages: C, assembler.

Architectures: 8051, PIC, 68000, 68HC11, 6805, V8, Z80, 8086, H8/300.

Personal: Very easy-going, undemanding but highly supportive of co-workers. Not a self-starter, I prefer to work in a team, small or large. I'm tireless and will work 24 hours a day if required, even skipping lunch breaks.

Interests: Wow, where do I start - I've been involved in rocketry, space, entertainment, defence, railroads, motor vehicles, domestic appliances, toys, you name it!

Goals: I want to be on your desktop. I want to make your application sing. I want to bang your bits, clear your RAM and crunch your calculations. Just give me a chance - I promise you won't regret it!!


```

RETURN
...
;--#--
;Begin dynamic scope of P'
;{Page = P' = 1}
CALL imm IncRegB
; Point A
;{Page = P' = 1}
;End dynamic scope of P'
;--#--
...
;--#--
;Begin dynamic scope of P'
;{Page = P' = 2}
CALL imm IncRegB
; Point B
;{Page = P' = 2}
;End dynamic scope of P'
A<-X reg CounterReg_P2
;--#--
...

```

The trick here is to introduce a symbol, *P'*, which represents the current page at some time during the execution of the program. *P'* is not only different in different places in the program, but different in the same place (in the subroutine *IncRegB*) at different times during the execution of the program. This is why the scope is called *dynamic*. Before *CALLing* *IncRegB*, we assert the beginning of the dynamic scope of *P'*. Then we define *P'* to be the current page. We *CALL* *IncRegB* with *{Page = P'}*. *IncRegB* doesn't change *Page*, so after *RETURNing*, we still have *{Page = P'}*. *P'* contains the value of *Page* before the *CALL*, *P'* doesn't change during the subroutine, and *Page* is *P'* after the *RETURN*, so we can assert that *Page* after the *RETURN* is the same as *Page* before the *CALL*. Certain precautions must be taken: the flow of control must encounter *Begin dynamic scope of P'* before it encounters *End dynamic scope of P'*, and between any two *Begin dynamic scope of P'*s in any possible flow of control, it must encounter an *End dynamic scope of P'*.

Now let's examine how this can be used to create more complex, more

useful subroutines. For example, *UpdateChecksum* could be rewritten as shown in Listing 2.

Note that *rE* is unpagged; therefore, it can be referenced without knowing exactly which page is active at the time. This new logical structure allows us to write subroutines that need a particular register page to be active, yet can be called from anywhere in the program without disrupting the active page at that point in the program. It is up to the programmer, of course, to ensure that *rE* does not become corrupted. If a second subroutine were to try to use this same trick of preserving the active page with register *rE*, and the second subroutine *CALLed* *UpdateChecksum*, then *rE* would be corrupted before the *RETURN* of the newer subroutine.

The ideal implementation of this form of paging management would be a stack. Upon entering a subroutine, the current page would be added to the stack. The page may be manipulated as needed in the body of the subroutine. Immediately before *RETURNing* from the subroutine, the old value of *Page* should be retrieved from the stack.

Reset and interrupt vectors

Every processor has a *reset vector*, a place where the processor starts executing instructions when it is reset (for instance, immediately after it receives power for the first time). Sometimes this is a fixed location in program memory, sometimes some memory is used to hold the address of the reset vector.

The processor may be designed to start in a known page, or it may not. If the active page after a reset is unknown, then the comment at the address vector must be *#####*. If the active page is set to 0 after every reset, the comment at the reset vector must have a *#* in the first character of the comment.

You could think of a piece of code (not stored in program memory) like this:

ImaginaryCode:

```

;{Page = (possible values of
Page on reset)}
GOTO imm ResetVector

```

This code is nowhere in the program memory, but it could be thought of as implicit code in the processor itself.

If a processor has interrupts, it usually has instructions for enabling and disabling the interrupts. I shall add a new bit type to the pseudo-assembly language, called *INTE* (Interrupt Enable). Interrupts are enabled by the instruction:

```
X<-1 bit INTE
```

and disabled with the instruction:

```
X<-0 bit INTE
```

A processor with interrupts also has one or more interrupt vectors. Some have automatic context saving, while others do not. Some of those that have automatic context saving may even set the active page to some known value when entering an interrupt. In any case, there is usually an instruction to return from interrupts. For this pseudocode, let that instruction be *RETINT*.

If the processor has automatic context saving that includes storing the value of *Page*, and *Page* is set to some fixed value when the interrupt service routine (ISR) is called, matters are simple. Start the ISR with a comment indicating the proper page, and use *RETINT* to finish the ISR.

If the interrupt doesn't use any paged registers, this situation is also easy. You should never write to *Page* in such an ISR. The interrupts don't even need to be considered for paging tracking, though a comment to this effect should be put in the source code. If any future modifications to the code require access to paged variables during the ISR, this comment could save a lot of time that would otherwise be spent examining the code

Change the way you think.

Think what you could change.

BUILD A MORE RELIABLE WORLD™

START WITH A MORE RELIABLE OS



It goes against conventional thinking. But to build more products in less time — without sacrificing reliability — you need more than good tools. You need a better OS.

An OS with an architecture so advanced it lets you pinpoint memory faults in minutes.

Prototype without system rebuilds. Add features without retesting your entire software suite.

Upgrade systems without rebooting. And respond faster to what customers want.

Now imagine. What if this same OS was backed by two decades' experience in the embedded market. An outstanding reputation for technical support.

And an OS company as reliable as its technology. Think how that could change your world, and your customers' world.

For the better.

www.qnx.com



Call about our eval system:

800 676-0566 ext. 2321

See us at Embedded Systems Conference Spring booth 1111

QNX®
RTOS



Build
a more
reliable
world™

Partial adherence [to formal methods], though academics may cringe, can be effective—especially if it is applied in critical areas, where a greater proportion of the benefits can be reaped with a smaller proportion of the drawbacks.

before modifications and/or testing and debugging the new code after modifications.

If the processor has automatic context saving that includes storing the value of **Page**, but does not set **Page** to some fixed value, you have to be careful with the beginning of your ISR. Any page that may be active at any time the interrupt is enabled may be active at the beginning of the ISR. Note that if interrupts are enabled when the processor is reset, any page which may be active on reset may also be active at the start of any ISR. Thus, if the page is unknown on reset and interrupts are enabled on reset, an ISR may start with any page active. On the other hand, if it is known that only one page is active while the interrupt is enabled, the interrupt must start with that page.

If the processor's context saving does not include **Page**, you have to be careful with the beginning of your ISR (as in the previous case) as well as with the end of your ISR. If only one page is supposed to be active when the interrupt service routine is called, then that page must be the only one active just before the **RETINT**.

The general case for handling paging with interrupts, when there is no automatic context saving, uses dynamic scope, like I described for subroutines. The current active page should be stored in an unpagged register immediately upon entering the interrupt service routine. Immediately before **RETINT**, the value should be restored. A particular snafu occurs when using this technique with interrupts. As I've discussed, the same register cannot be used to store the page in two different subroutines when one subroutine calls the other. (This also means that the same register cannot be used by both an ISR and a subroutine that is executed when interrupts

are enabled.) Similarly, if the processor has low-priority and high-priority interrupts, the same register cannot be used in one of both types of ISR. If a low-priority interrupt starts using a register, and a high-priority interrupt occurs, the high-priority ISR will begin and finish executing before the low-priority ISR gets any further. In this way, that very important register could become corrupt.

If resources in the processor allow it, you could use a stack, too. Push the current page at the beginning of the interrupt service routine, and pull it just before **RETINT**.

Semi-formalism?

<SERMON>

This article is inspired by formal programming methods. Total adherence to formal methods requires well trained programmers, and would make system performance extremely well defined, with absolutely no bugs. It would also mean that there would be a long, complicated mathematical expression (pages long in most cases) distributed with each program—and it would probably be indecipherable to any businessman or marketer. This is a good academic ideal, but it isn't practical for embedded systems design. (Not enough research has been conducted in formalizing such aspects of programming as real-time operation.) Neither is this ideal marketable (except to computer science and mathematics experts, who could understand and appreciate the formal specification).

Partial adherence, though academics may cringe, can be effective—especially if it is applied in critical areas, where a greater proportion of the benefits can be reaped with a smaller proportion of the drawbacks. I have found paging to be just such a critical area. Consistency, ease of

maintenance, development speed (once the developers get accustomed to it), and fewer bugs (to the point of none at all, depending on how far you take it) are some of the benefits that come directly from the use of formal methods. These benefits go to the developers, the project manager, the people who rely on the program, and the programmers who need to add a new feature five years down the road. The drawbacks involved in following these guidelines are a requirement for discipline (though hardly approaching the discipline required for full formality), and probably a slower initial development speed.

</SERMON>

This article is meant as a guide to getting your paging right. It is not a complete guide for all processors. For example, many processors have indirect register addressing, an issue I haven't touched on here. My guide may be used as a guarantee of correct paging if such things as indirect addressing are not used, or as an aid to keep paging correct if those features are used.

Other topics I haven't covered in this article include assembly instructions such as **GOTO reg rA** (or **X<-A reg PC** where **PC** is the program counter), and even **CALL reg rA**. Part of the reason I've avoided these subjects is that I consider the use of instructions such as these to be bad programming practice, and so never use them myself. I haven't covered the **Page<-A** instruction where **A** has an unknown value for the same reason—I consider that notation to be poor programming practice too.

I hope the method I've recorded here will help other assembly language programmers. These practices have proven themselves very effective in my experience.

esp

Hugh O'Byrne graduated from University College Dublin in 1995 with a BS in computer science. He's currently pursuing an MS in mathematics.

✓ Royalty Free

✓ Complete Product line

✓ Focus on Service



All You NEED in an RTOS

www.atinucleus.com

There is
only one conclusion.



Getting an embedded product to market requires core elements. Once these prime responsibilities are accounted for, you are free to do what you do best: Develop.

Accelerated Technology provides embedded developers with all the established and tightly integrated Real-Time Operating System software. Our complete embedded product line is built on the Nucleus PLUS real-time kernel with many complementary products developed in-house. Some of these products include: Nucleus MNT, a Windows-based prototyping environment; and Nucleus EDE, an embedded development environment based on Microsoft Developer Studio™; Nucleus PLUS, a real-time kernel; Nucleus NET, a TCP/IP protocol stack; and Nucleus UDB, a portable source level debugger. With the combination of SurroundView and Nucleus ProView, Accelerated Technology is leading the embedded market with best-in-class profiling tools. In addition, Nucleus WebServ completes the application development process by providing an embedded web server.

Accelerated Technology supplies Nucleus embedded software with complete source code and without charging royalty fees. Therefore, we are able to provide you with a comprehensive product line with less expense than creating it yourself.

We are unique in our focus on customer service. It is our promise that Accelerated Technology will do what it takes to maintain our well-known position as a company that is responsive to the needs of developers.

Accelerated Technology provides a complete line of royalty free software while maintaining a focus on customer service. All of these benefits from one RTOS. There is, indeed, only one conclusion, Accelerated Technology, Inc.

Contact Information: 720 Oak Circle Dr. E. Mobile, Alabama 36609
Phone: 800.468.6853 - Email: info@atinucleus.com - Contact: Sales Dept.



All You Need in an RTOS. Royalty Free.

PHILIPS



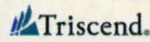
AMD

ARM

NEC



ARC



Hands-on Computer Labs in Embedded Systems Applications, Chips & Tools!

SIMPLIFY THE DESIGN PROCESS AND GET YOUR BREAKTHROUGH PRODUCT TO MARKET QUICKER!

This is your chance to evaluate dozens of the leading embedded microcontrollers and their development tool chains—in an actual college laboratory environment!

PORTABLE
Design

Embedded Systems



April 3-7, 2000

Mission College,
Technology Center
Santa Clara, CA.

A FULL WEEK OF 2-HOUR LABS & TUTORIALS ON THE HOTTEST NEW EMBEDDED APPLICATIONS:

Embedded Internet, Industrial & Motor Control, USB, Communications, Routers, Information and Appliances, CAN, etc.

PLUS: • WindowsCE Workshops • Panel Discussions on EEMBC, NEXUS, System on a Chip, etc. • Discussions on Trends in Embedded Systems Development by Jerry Krasner • Analysis by Tom Starnes and Jim Turley on the newest microcontroller trends • and more!

Applications

Take Labs & Tutorials on THE hot applications, including:
Embedded Internet
System Integration
Universal Serial Bus
Wireless
Motor Control
.....and dozens more!

Special Topics

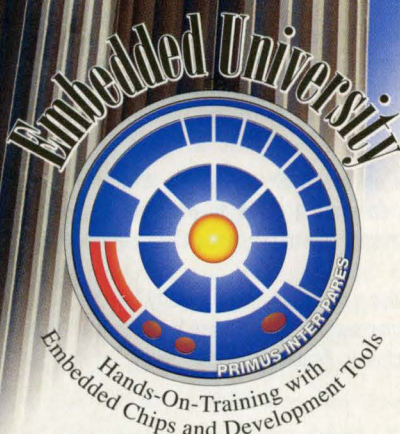
Select from daily seminars and panel discussions on:
Programming of Flash Micros
OPAMPs Embedded in 8-Bit Micros
Benchmarking 8-Bit Compilers
Debugging Motorola MCU Designs
Writing Software for ARM Designs
Create your own CSOC
Selection of Low-Cost MCUs
And more to choose from!

Chips

AMD 186, Elan, and K6
Infineon Technologies C500, C166, TriCore
Philips 8051, XA
STMicroelectronics ST6, ST7, ST9, ST10, STPC
ARM All Families
Zilog Z8
Mitsubishi M16C, Slim 740
NEC V800, 78K
Triscend E5 Configurable SOC
ARC Cores Ltd. 32-Bit Processor
Analog Devices Microconverter
National Semiconductor COP8
Microchip PIC 16
Hitachi H8
Motorola 68HC08, M-Core
.....AND MORE!

Tools

Development Tools from the leading suppliers worldwide presented together in tool chains!
*Compilers
*Assemblers
*Debuggers
RTOS
*Emulators
*IDEs
*Logic Analyzers
...and Dozens of tools designed for special applications
Work with the experts in computer labs designed specifically for your needs!



Register On-line Today and Receive FREE Evaluation Boards, a Visor Delux or PalmV

www.EmbeddedU.com



Don Morgan

A Paean to Noise

For the last several columns, I have been designing and illustrating filters based on simple and familiar concepts in an attempt to demystify the world of signal processing. While researching this month's column, I was re-reading part of Sophocles Orfanidis' wonderful book *Introduction to Signal Processing*.¹ In it, I found a statement that caught my eye: "One of the most important applications of DSP is removing noise from noisy signals." What a thought. It occurred to me, however, that I had not stopped to put much definition to the term "noise" in all these months. Orfanidis' statement is, without a doubt, true. But without qualification, it can lead to any number of philosophical conclusions.

I suspect that most people feel that noise is something you know (and hate) intuitively, something like that awful fly that will not leave you to your hamburger and potato salad. It's that nasty stuff on the radio you hear when you get too far from the station; it's snow on a television screen; it's 20 or 30 kids with flutophones performing at a PTA meeting; it's that stuff causing your wrong A/D conversions; it's the result of quantization; and so on. What causes noise? It might be a 60Hz hum; that block of integrated gate bipolar transistors switching hundreds of Amps in nanoseconds; arithmetic; or even souls that have passed over to the other side. And you get rid of noise with good bypassing; with ferrite beads; by adding as many 0.1 microFarad capacitors as you can without completely swamping out your signal; by filtering; or just by ignoring it.

In search of a definition, it's easy to say that noise is that part of a signal

that you don't want. And this is easy to prove, too—if you are processing signals at 20kHz, and find that energy from 50kHz is finding its way into your system, you would call that noise. Some people consider Mozart or Dr. Dre noise. Noise is, indeed, in the ear of the beholder.

But, perhaps you'll be surprised to find that the stuff so many call noise can be a useful tool. In fact, many of us

Correlation represents a correspondence among data. Actually, it's the search for relationships between data. Two sets of data may vary from identical to completely unrelated. A good deal of real and pseudoscience is built on this. If a doctor sees a 200-year-old man eating a celery stalk, he may put those two data together and conclude that vegetarians live longer. In more rigorous studies, correlation

One man's music is another man's noise. This month's column describes several types of noise that may be music to a signal processor's ears.

spend some time creating noise for the purpose of analyzing systems and creating effects. And not just any noise—specialized noise sources can, in fact, require a great deal of effort to create.

So, I thought it might be appropriate to add some definition to the subject of noise, not only for analytic purposes but also for its value in synthesis and control. In this column, I will define some different kinds of noise.

Characterizing noise

Since little difference exists between noise and other data, aside from perspective, it's not surprising that we use the same tools to characterize and define noise as we do to characterize and define all other signal data, particularly autocorrelations and power spectra.

is a measure of how data group together, and involves the development of a mean and deviation. Mathematically, one calculates a correlation in the same way one does a convolution without the time reversal (note the interesting implications this has for symmetrical FIR filters):

$$C_{xy}(t) = \frac{1}{\sigma_x \sigma_y} \int_{-\infty}^{+\infty} x(\tau)y(t+\tau)d\tau \quad (1)$$

Applying this to two data streams, we can find the areas of correspondence in the two signals.

When we take the autocorrelation of a signal or sequence, we are performing this correlation on a signal with itself. The results of this process are equal to a Fourier transform of the power spectral density, that is, the squared magnitude of the transform:

$$C_{xx}(t) = \int_{-\infty}^{+\infty} y(\tau)y(t+\tau)d\tau$$

$$= \int_{-\infty}^{+\infty} |Y(f)|^2 e^{j2\pi ft} df \quad (2)$$

We can use this data to determine where the energy in a signal is. If you decide that one or more of the peaks in

this analysis is unwanted, or "noisy," you may apply a filter to the data to remove it. For instance, a 60Hz "noise" is found everywhere (in North America). An autocorrelation performed on a signal with this noise will find energy concentrated at 60Hz, in addition to the energy in the desired spectrum.

All noises are not the same

Many people consider all noise to be the same and as a result, they have a hard time getting rid of their noise problems. The previous formulae provide a means to define that unwanted signal. If your objective is to eradicate noise, you'll find that some solutions exist for different groups of noise, but no one solution works for all noise.

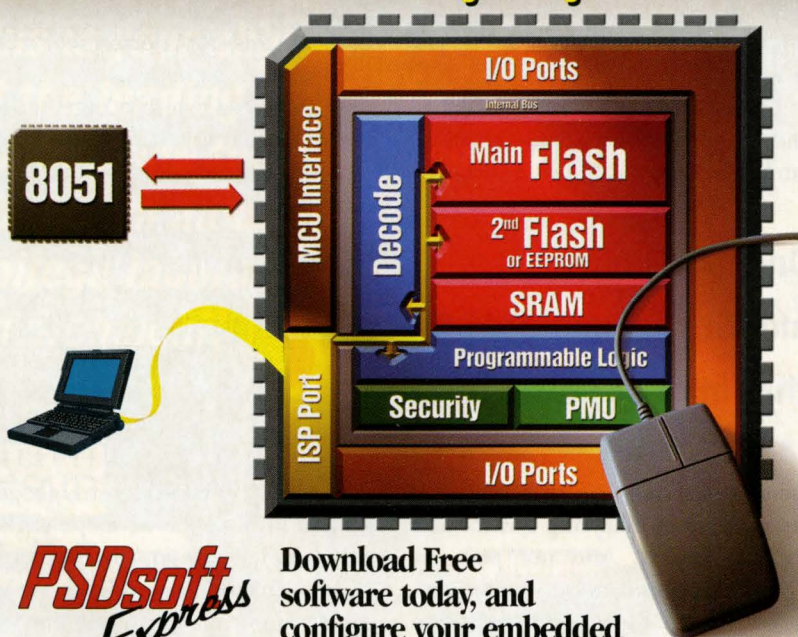
Most of you are aware of the IIR and FIR filters that are common in signal processing. The purpose of these filters is to shape or define the spectrum in a certain region, but we often use them to remove an unwanted signal. Unfortunately, this, like 0.1 microFarad capacitors, won't work for everything. In video work, for example, applying such a filter without blurring the signal and losing edges is difficult. Still, noises find their way into the signal and annoy the viewer—high-frequency noise that can clutter the signal with bad pixels. A median filter can be used to remove the offending bits. This filter finds the median value over a small region and uses it to replace the central pixel. To do this, the pixel values over the region are collected into an array and sorted. The value in the middle of the array is the median. The size of these arrays may vary, but are usually relatively small (five to 13 pixels). This technique may be used to eliminate other sources of erratic noise, and this filter may be applied in more than one dimension.

The biggest noise of them all

The most well-known noise is probably white noise, yet another type of noise that doesn't yield easily to simple high- or low-pass filters. This is a noise whose energy per frequency is constant throughout the specified spectrum. White noise in your system exists with an equal energy at every frequency. Neither low passing nor high passing your signal will deal with this noise effectively, since it exists in all the bands.

An interesting thing about white noise is that humans are quite familiar with it and adept at digging information out of it. It is even reputed to have heal-

In-System 8051 Memory System



PSDsoft
Express

Download Free software today, and configure your embedded design with your mouse!

Waferscale's new development and programming software... PSDsoft Express™, provides an easy point and click environment for configuration of **EasyFLASH™** PSDs. It guides you through an entire design from configuring the MCU interface and selecting a PSD, to programming the code and firmware into the PSD. All in less than 2 hours! Now you can design Flash, SRAM, additional I/O, and advanced ISP capabilities into your next embedded design by adding a single chip, in a single afternoon. Visit the URL below to download your FREE fully functional copy of PSDsoft Express today!

www.waferscale.com/dlexpress.html



Waferscale-USA
Tel. 800-832-6974
Fax 510-657-5916
info@waferscale.com

Waferscale-EUROPE
Tel. 33-1-69320120
Fax 33-1-69320219
wsifrance@wsiusa.com

Waferscale-ASIA
Tel. 82-2-761-1281
Fax 82-2-761-1283
james@mail.wsiasia.co.kr

\$35
Value!

Free Pass

to the Product Exhibits

Bring this ad with you to see:

Three-Day Admission to Product Exhibits

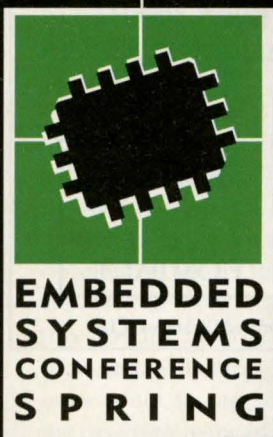
February 29, 12:00–7:00pm

March 1, 10:00–6:00pm, March 2, 10:00–4:00pm

More than 170 leading companies featuring:

- Embedded Internet Tools
- DSPs and RTOSes
- Development Tools
- System Design Tools
- Networking Tools

Get your hands on all your product options and talk face-to-face with the “factory people” who design your current favorites. You’ll get the low-down on compatible products and be the first to hear about what’s coming down the pike.



Keynote Address:

Clifford Stoll: “Stalking the Wily Hacker”

February 29, 10:30–11:30am

Back by popular demand, Clifford Stoll, industry celebrity and best-selling author of *The Cuckoo’s Egg*, returns in what will surely be another frenetic, unpredictable account of hackers, technology, and the “cult of computing.”



Special Guest Lecture

James Rumbaugh: “Building the Next Generation of Real-Time Embedded Systems with UML”

March 1, 6:00–7:00pm

Hear the chief developer of the object modeling technique (OMT) and the co-developer of Unified Modeling Language.

Panel Discussion

“The Open-Source Phenomenon:
Is the Embedded Systems Business Ready for It?”

February 28, 6:00–7:30pm

Special Leap Year Celebration

“Leap Into the Caribbean Zone”

February 29, 7:00–9:00pm

Win a trip for two to the Bahamas at our tropical celebration. Additional prizes will be awarded in the limbo and “tacky tourist” contests.



Show Floor Reception

“A Whole New Ballgame”

March 1, 5:00–6:00pm

February 28–March 2, 2000
McCormick Place South
Chicago, IL

Bring this with you or register
today to avoid on-site lines!

www.embedded.com/spring

800-789-2223

CMP

Embedded Systems
PROGRAMMING

CMP

EE TIMES

CMP

DOS COMPATIBLE FILE SYSTEM

Since 1987 in Hundreds of Applications Worldwide

Selected as the File System for Several
Major Embedded Operating Systems

DOS/Win95/FAT32 Compatible

Simple OS and CPU Porting Layer

Contiguous File Support

Realtime Extensions

Includes Support for IDE, Floppy, ROM/RAM Disk,
PCMCIA, Compact Flash

CDROM Support Available

100% 'C' Source Code • Royalty Free

Still Only \$3825⁰⁰



Visit Our Web Site at:

www.etcbin.com

TOLL FREE **1 800 428-9340**

Outside U.S. Call 978 448 9340

email: sales@etcbin.com

Dinkumware, Ltd.
Genuine Software

www.dinkumware.com



Dinkum

C++

Library

POWERFUL

*What size footprint does
your software project need?*

Dinkum

Abridged

Library

STRONG

Dinkum

EC++

Library

COMPACT

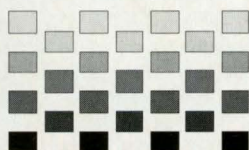
Dinkum and Dinkumware are registered trademarks of Dinkumware, Ltd.

32/64-Bit CPU Development Boards

- ARM
- MIPS
- Power PC
- M-Core
- PCI

**Cogent Modular
Architecture
(CMA)** - a series of
flexible development
platforms offering
advanced on-board I/O,
PCI expansion and
interchangeable CPU
Modules for a variety of
32/64-Bit architectures.

CMA
**COGENT
MODULAR
ARCHITECTURE**



COGENT

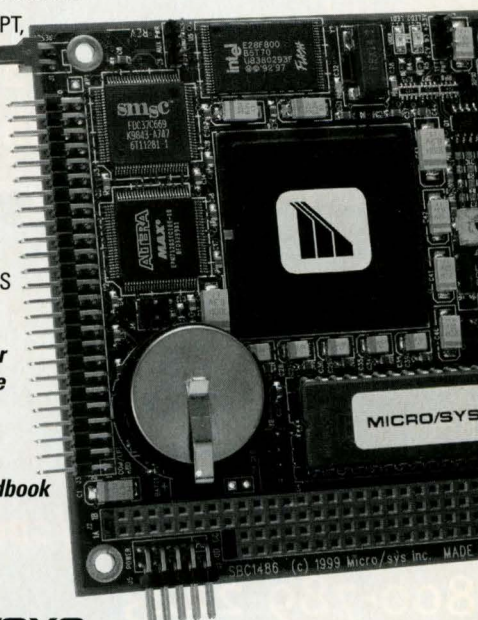
Cogent Computer Systems, Inc.
Tel: 508-278-9400 • www.cogcomp.com

Action Packed 486's

Look what you can have on a PC/104-sized SBC

- 486DX and CRT/LCD VGA
- 64M RAM and 72M Flash
- COM1, COM2, LPT,
KBD, mouse,
touchscreen
- On-board suite
of firmware
including
DOS emulator
- Boot Dos,
Win98, Win,
CE, LINUX, RTOS

*Call or visit our
website for the
SBC1486
data sheet or
a FREE
284-page Handbook*



MICRO/SYS

Montrose, CA

(818) 244-4600 • Fax: (818) 244-4246

www.embeddedsys.com

ing powers—some physicians (medical doctors and dentists) are experimenting with it for use as a light anesthesia.

By definition, white noise doesn't correlate because it is random; therefore, it has no bias. As a result, white noise can be added to signals and arithmetic (and is all the time in your A/D converters) without adding long-term error—it is sometimes used to extend eight bits to 16 bits. With proper handling, white noise may also provide the energy for creating new sounds, from individual voices to nature sounds to many of the other noises I'll introduce next.

An FFT or wavelet (sub-band) filter system is effective in exposing white noise and thresholding the results before application of the inverse transform can remove it. However, white noise is so useful for analysis or in the creation of effects that many people are hard at work creating it! To do this, they use random number generators. And, as it turns out, good white noise is not so easy to produce—as is the case with many of the other noises we will discuss shortly.

The problem with creating the white noise you desire is that writing a truly random random-number generator is difficult—sooner or later, the sequence wants to repeat. Often, such a generator requires a seed. Given the same seed, some generators will give you the same sequence. A number of treatises are available regarding this. One of the best is in D.E. Knuth's *Seminumerical Algorithms*.² Another well known source comes from *Transactions of the ACM*: "Random Number Generators: Good Ones are Hard to Find" by S. K. Park and K. W. Miller.³

A search of the Web produces a number of random number generators, too many to mention, that you may use to make noise locally; there is even a Random Number home page at www.npac.syr.edu/projects/random.

Colors of noise

White should embody all colors and, therefore, white noise should contain


all forms of noise. Indeed, white noise is defined as having a power density that is constant over a finite frequency range. This suggests that other "colors" of noise exist, and that proves to be true. Here is a short listing of popular noise colors with their definitions:⁴

- Pink noise. Power density decreases
- Red noise (oceanographic defini-

3dB per octave with increasing frequency (density proportional to $1/f$) over a finite frequency range, which does not include DC. Each octave contains the same amount of power. Producing 3dB per octave isn't easy; ripple-free pink noise is hard to find

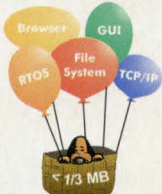
smx[®] MODULAR RTOS

SYSTEM SOFTWARE OUTFITTERS




SMX/PEG Evaluation Kit

Includes **smx**, **smx++**, **PEGmt** multi-tasking GUI, and **WinBase** to compile, run, and debug in Microsoft Developer Studio.




Multitasking for x86, PowerPC, and ColdFire




smx. Full-featured, fast, small, preemptive kernel. Supports all x86, PowerPC, and ColdFire processors. x86 support for real mode (w or w/o DOS), 16-bit seg. prot. mode, and 32-bit flat prot. mode. 32-bit works with **pmEasy32**, Win95, 98 & NT. Dynamic load module support.

Networking




smxNet. TCP/IP. No copy. ICMP, ARP, BOOTP, Ethernet, SLIP, and PPP drivers. FTP, NFS, Telnet, SNMP, DHCP, Web server, HTML v3.2 Graphical Browser, SMTP/POP3 email.

x86 Protected Mode




pmEasy. 16- or 32-bit protected mode entry, DPMI services, application loaders, Soft-Scope[®] and VisualProbe[®] debugger support.

Debugging




smxProbe. Provides tracing and task debugging. **smxAware** & **DEBUG/smx** add smx-awareness to debuggers. **WinBase** permits IDE development & debugging at low cost.

DOS-Compatible File System




smxFile. Full-featured file manager. IDE, floppy, flash, RAMdisk, PCMCIA, LS120, and SCSI drivers available. **smxCD**. ATAPI CDROM file system.

C++ Classes




smx++. Class library built upon smx. Provides complete support for embedded C++.

User Interface




PEG. Portable Embedded GUI. **PEGmt** adds multitasking. Support for **SciTech MGL**, **MetaWINDOW**, and **Zinc** graphics. **smxWindows**. Text windowing library.

DOS & Windows Emulation



X-DOS. Full v5 DOS. **unDOS**. DOS, BIOS, and Windows emulator. **EXE Bootloader** boots from BIOS.

Tool & Board Support from A-Z



AMD	MetaGraphics	SciTech
Beacon	MetaWare	SDS
Borland	MetroWerks	Transvirtual
Diab	Microsoft	VersaLogic
EBS	National	Watcom
Intel	Paradigm	Zinc

NO ROYALTIES • 6 MO FREE SUPPORT • 30-DAY FREE TRIAL

MICRO DIGITAL 1-800-366-2491 www.smxinfo.com/rses.htm



DSP WORLD SPRING DESIGN CONFERENCE

San Jose McEnery Convention Center
April 10-12, 2000

DSP World Spring Design Conference offers practical training for the specialized knowledge and skills needed to implement DSPs in embedded systems. More than 35 classes, all taught by experts in the field, will be offered at **DSP World Spring Design Conference**.

Classes include:

- VoIP: Fundamentals and Applications
- RISC-DSP: An Exploration into the How's and Why's
- Design of Fixed-Point Systems - Advanced Methodologies
- An Introduction to DSP

DSP World Spring also features two days of product exhibits where leading manufacturers will showcase hundreds of DSP products and tools, including:

- microprocessors
- DSP boards
- operating systems
- algorithms

**Request your
FREE catalog
today!**

phone: TOLL FREE 1.800/789.2223
415/278.5231 (attendee info)
415/278.5313 (exhibitor info)
fax: 415/278.5390
email: dspworld@mfi.com
web: www.dspworld.com

produced by:



co-sponsored by:



www.dspworld.com

tion). This is oceanic ambient noise. It is red due to the selective absorption of higher frequencies

- Orange noise. This is quasi-stationary noise with a finite power spectrum and a finite number of small bands of zero energy dispersed throughout a continuous spectrum. These bands of zero energy are centered about the frequencies of musical notes in whatever system of music is of interest. Since all in-tune musical notes are eliminated, the remaining spectrum could be said to consist of sour, citrus, or "orange" notes
- Blue noise. Power density increases 3dB per octave with increasing frequency (density proportional to f) over a finite frequency range. This can be good noise for dithering
- Purple/violet noise. Power density increases 6dB per octave with increasing frequency (density proportional to f^2) over a finite frequency range
- Gray noise. Noise subjected to a psycho-acoustic equal loudness curve (such as an inverted a-weight curve) over a given range of frequencies, so that it sounds like it is equally loud at all frequencies
- Brown noise. Power density decreases 6dB per octave with increasing frequency (density proportional to $1/f^2$) over a frequency range, which does not include DC. The name does not come from a color but is actually a corruption of Brownian motion. Also known as *random walk* or *drunkard's walk* noise
- Black noise (silent).
 - 1) Any of the output of an active noise control system that cancels an existing noise
 - 2) Has a power density that is constant for a finite frequency range above 20kHz. This is something like ultrasonic white noise. This black noise is like the "black light" with frequencies too high to be perceived, but still capable of affecting you or your surroundings
 - 3) Has an f^{β} spectrum, with $\beta > 2$, and according to lore, embodies the

characteristics of catastrophes both natural and unnatural such as floods, droughts, bear markets, and outages

Bringing more noise

In signal processing, we often refer to the *Dirac function* or *unit impulse*. This impulse is a device with essentially zero width and infinite height. If we hit an object with this impulse, whether it is an atom or the Asian continent, this impulse is short and powerful enough to set that object ringing at its natural frequency without dulling, blurring, or otherwise becoming involved in that ringing at all.

Infinitely short and infinitely tall impulses are hard to find. But it's possible, with varying degrees of difficulty, to produce signals that embody selected bandwidths and selected power densities and then apply these signals to objects to see what part of the signal

that object absorbs or resonates to.

Next month, I'll continue discussing applications for noise and will present some code that you may use to produce your own noise. **esp**

Don Morgan is senior engineer at Ultra Stereo Labs and a consultant with experience in signal processing and related industries.

References

1. Orfanidis, Sophocles. *Introduction to Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1996.
2. Knuth, D.E. *Seminumerical Algorithms*. Reading, MA: Addison-Wesley, 1980.
3. Park, S.K. and K.W. Miller. "Random Number Generators: Good Ones are Hard to Find," from *Transactions of the ACM*, November 1988.
4. Much of the data for the definitions in this section came from "Colors of Noise" by Joseph Wisniewski, available at www.msaxon.com/colors.html.

KEIL SOFTWARE

Compilers, Assemblers, Debuggers, and RTOS for the 8051, 166, and 251

Listen to what our customers are saying...

"Thanks to you for your time and help. I was able to rebuild my ROM this weekend and have everything ready for Monday" *E. M.*

"GREAT web site. Easy to navigate and lots of useful information." *R. L.*

"Get on the horn with Keil's support team... it won't cost you a cent... They won't bite your head off for asking questions." *M. C.*

"The free evaluation CD, the toll-free technical support telephone number, and the availability of people to speak with were all key factors in our decision to purchase your products." *D. C.*

"This (support solution) was exactly what I needed! Thanks! :-)" *R. S. N.*

"We tried what you suggested. Now, it works. I really appreciate your efficient support!" *X. J.*

"I couldn't find our lab copy of the manual, so I used your (web) search engine to find the syntax for the directive. It was quicker than looking for the manual !!!!!" *N. M.*

"I LIKE my Keil tools..." *B. N.*

C51

C51 is the industry standard ANSI C compiler for the 8051. It supports all 8051 derivatives including the new USB & CAN devices.

C166

C166 Version 4 is the easiest toolset to use for C16x code development. It seamlessly integrates with all C16x and ST10 emulators.

C251

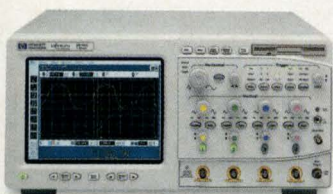
New compiler technology built into C251 is designed specifically for the 251 architecture. It takes full advantage of the new 16 and 32-bit registers and instructions.

Call Today for a free CD-ROM.

Keil Software, Inc.
16990 Dallas Parkway, Suite 120
Dallas, TX 75248-1903

Sales: **800-348-8051**
972-735-8052
FAX: 972-735-8055

www.keil.com



Infiniium. Because some things you should only have to learn to do once.

from \$9,995*

Agilent Infiniium Oscilloscopes

- 500 MHz, 1 GHz and 1.5 GHz models
- Up to 8 GSa/s sample rate
- 2- and 4- channel models
- Advanced triggering
- Communication mask testing
- Voice control (English only)

The idea is to debug your design, not to figure out how to use your scope. High performance doesn't have to mean difficult to fathom. Not with an Infiniium oscilloscope from Agilent Technologies.

All those soft keys and menu layers have been replaced with something more engineer friendly. There's an analog-like front panel that's straightforward and simple (the green knobs control the green waveform on the screen, for example). The Windows 98® interface handles advanced measurements with drag and drop ease. And LAN connectivity makes sharing your work easy.

If you need communication masks for compliance testing, Infiniium offers those, too. And should you have questions only another engineer can answer, you can call one on our toll-free number.

So even if you're a fan of someone else's scope, you might want to discover first-hand how good an Infiniium really is. A free, no-obligation demo makes that a lesson easily learned.

www.agilent.com/find/Infiniium1

1-800-452-4844,** Ext. 6757



Agilent Technologies
Innovating the HP Way

*U.S. list price.

**In Canada, call 1-877-894-4414, Ext. 6447. ©1999 Agilent Technologies ADGPP003/ESP
Windows 98 is a registered trademark of Microsoft Corporation.



Tools for Embedded Developers

Software

Software for automotive ECUs

Rhapsody in MicroC is part of the family of Rhapsody Visual Programming Environments, focused on automotive market software development needs. Rhapsody in MicroC is specifically developed for automotive electronic control units (ECUs). It graphically captures the software architecture and its behavior, generates fully readable, deployable code, and validates the code up front before anything is built. Rhapsody in MicroC is available now for Windows NT host operating systems.

I-Logix

Andover, MA
(978) 682-2100
www.ilogix.com

Developers kit for DSP

The Motorola DSP Developers' Kit simulates and tests algorithms for Motorola's 56300 and 56600 families of fixed-point DSPs. It integrates the DigitalDNA development environment with the Simulink and MATLAB DSP design tools. With the kit, developers can perform system-level design, DSP implementation, and verification of production code in one design flow. The DSP Developer's Kit is available now for PCs running Windows NT, Solaris, HP-UX, and Windows 98 platforms will also be supported. The kit starts at \$995 for an individual PC license.

The MathWorks

Natick, MA
(508) 647-7000
www.mathworks.com

Motorola

Austin, TX
(512) 502-2100
www.motorola.com

Application monitoring tool

The SurroundView application monitoring tool is now integrated with the Nucleus PLUS kernel for Motorola Computer Group's MBX boards. SurroundView for Nucleus enables developers to monitor the dynamic behavior of embedded applications by tracing execution. It enables data monitoring and profiling capabilities, all displayed in graphic form. Developers can detect dynamic misbehavior of applications, isolate the problematic variables, and pinpoint the erroneous code in real time, without halting the application. SurroundView for Nucleus supports Ethernet TCP/IP and serial target connections. It's available for Windows NT and Windows 2000 host systems for \$2,995 per seat.

Accelerated Technology

Mobile, AL
(334) 661-5770
www.atinucleus.com

IDE for DSP

SystemView TMS320C6000 DSP Design Suite is a DSP development tool that supports the eXpressDSP Real-Time Software Technology. It provides an integrated design environment that offers bit-true fixed- and floating-point DSP system design, C code generation, and integration with Code Composer Studio for prototyping, real-time analysis, and debugging. The toolsuite includes three components: SystemView for system design/simulation/

analysis, the SystemView DSP library of fixed- and floating-point blocks, and the new Real-Time DSP Architect, which provides C code generation and an interface to Code Composer Studio. SystemView DSP Design Suite for Texas Instruments is available now for \$8,995. The Real Time DSP Architect—TI TMS320-C6000—can be purchased separately for \$4,995.

Elanix

Westlake Village, CA
(818) 597-1414
www.elanix.com

GUI ports to RTOS and processor families

Ports of Eyelet GUI are now available for five RTOS families and two processor families. Eyelet GUI is independent of desktop interface conventions, and not restricted to a single processor, compiler, or RTOS. Ports are available for the following RTOSes: pSOSystem, PreciS/MQX, μ COS, eCOS, and LynxOS. Eyelet GUI is being ported to the following processor families: ARM7 and PowerPC MPC821/MPC823. The royalty-free Eyelet GUI is available now.

MoJo Designs

Boulder, CO
(303) 443-5035
www.mojodesigns.com

Hardware

Evaluation hardware for networked devices

The Slim740 ChipConnect demonstration board is the newest addition to the ChipConnect family. The board, based on the EMIT (Embed-

ded Micro Internetworking Technology) architecture, lets developers design network-enabled products based on the Slim740 MCU. The evaluation hardware consists of two reference-design/demo boards, plus a mini-emulator for the mask ROM/one-time programmable Slim740. Software tools include IAR's compiler and debugger, and a 60-day evaluation version of EMIT. The basic Slim740 ChipConnect board contains a socket for a Slim740 MCU and a header for attaching the mini-emulator. The Light&Sound Add-on Board plugs into headers on the basic board and provides a four-digit alphanumeric display, as well as audio circuitry with a speaker. The development board was designed for the 36-lead SSOP version of the Slim740. The board and supporting software are available now.

Mitsubishi Electronic Device Group

Sunnyvale, CA
(408) 730-5900
www.mitsubishichips.com

In-circuit emulator

The Model SYS4K/opt34R is an in-circuit emulator for the 79RC64000 64-bit microprocessor family from Integrated Device Technology. The SYS4K ICE communicates with the CPU core using a dual-port ICEport memory system. It uses no target memory and is configurable to support a variety of target system memory architectures. Features include Ethernet and high-speed serial interfaces to the host computers; full C source-level debugger integrated with the emulator; unlimited software breakpoints and up to 50 hardware breakpoints; up to 8MB of optional overlay memory for debugging flash

and ROM-based code; backward-compatible support for other IDT 64-bit processors; and expandability to support R4K and R5K microprocessors from other MIPS licensees. The SYS4K is available now. Prices range from \$19,995 to \$33,400.

Embedded Performance Inc.

Milpitas, CA
(408) 957-0350
www.episupport.com

OCDS-level 2 emulator

The DProbeTriCore Level 2 emulator works with the TriCore 32-bit architecture from Infineon. The main benefits of the emulator are the OCDS-level 2 support via a dedicated emulation chip and ease of use through plug and play. The DProbeTriCore Level 2 comes with a 2,048K trace buffer and operates at up to 100MHz with 10ns

68HC05



68HC16

68HC08

NEW: VERSION 6.0!

Some things will never change.

First, Archimedes Software, Inc. C compiler and hardware emulator tools enable you to quickly develop stable, real-time applications based on Intel and Motorola microcontrollers.

And second, Archimedes customer support leads the industry in customer response time as well as qualified support engineers.

Archimedes Software, Inc. is the reason why tens of thousands of embedded designs ship on time and pack the most features. Find us at:

www.archimedessoftware.com

68HC12

Archimedes Software, Inc.
303 Parkplace Center #125
Kirkland, WA 98033
425-822-6300
425-822-8632 FAX

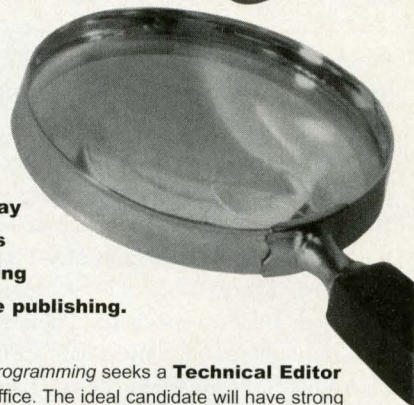
683XX

8051XA

8051/251

Looking for new Challenges?

If you have an engineering background and good communications skills, then you may have what it takes to enter the exciting world of magazine publishing.



Embedded Systems Programming seeks a **Technical Editor** for its San Francisco office. The ideal candidate will have strong software and hardware skills, design or applications engineering experience, the ability to translate technical information into fluid prose, and the ability to meet rigorous deadlines. The position entails writing and editing technical articles and product reviews, working with the Editor on technical aspects of the magazine and conferences, and meeting with vendors. Prior writing experience is a plus.

If interested, please send resume and cover letter (including salary requirements) to:

Miller Freeman Human Relations
525 Market St., Suite 500 • San Francisco, CA 94105
fax: 415.278.5341

bus cycles. Because it has its own instruction set, the built-in PCP (peripheral control processor) I/O processor can execute the program independently of the CPU. The DProbeTriCore Level 2 emulator is available now.

Hitex Development Tools

Sunnyvale, CA
(408) 733-7080
www.hitex.com

Chips

Eight-bit microprocessor

The Rabbit2000 microprocessor is an eight-bit processor with a C-friendly instruction set. On-chip peripherals include four serial ports; glueless memory and I/O interface; remote cold boot; on-board slave port; 40 par-



Rabbit2000 microprocessor from Rabbit Semiconductor

allel I/O lines; seven different timers; precision pulse generation hardware; battery-backable time/date clock; and watchdog. The Rabbit2000 is supported by Dynamic C software, which provides an interactive compiler, editor, and debugger. The Rabbit2000 features an updated 64180/Z180-style architecture. It's available at clock speeds up to 30MHz and comes packaged in a 100-pin PQFP. It costs \$8 to \$10, depending on quantity. A development kit including a single-board computer, a complete Dynamic C software system, a prototyping board, a power supply, and a PC serial interface cable is available now for \$139.

Rabbit Semiconductor

Davis, CA
(530) 757-8400
www.rabbitsemiconductor.com

Eight-bit flash MCU with migratable memory

The 28-pin PIC16F872 is the newest member of the PIC16F87X family of eight-bit flash microcontrollers. It uses migratable memory technology which provides socket and software compatibility among all equivalent ROM, one-time programmable, and flash memory MCUs in the PICmicro family. With 2K x 14 bits of flash memory and 64 bytes of EEPROM data memory, the chip features an operating voltage range of 2V to 5.5V. It also has a five-channel, 10-bit A/D converter; brownout detection; up to 5 MIPS performance at 20MHz; I²C or SPI communications capability for peripheral expansion; two eight-bit timers; and one 16-bit timer. Like the other PICmicro chips, the PIC16F872 features in-circuit serial programming, which allows the MCU to be programmed after being placed in a circuit board. The chip is supported by the MPLAB-ICD evaluation kit and by the MPLAB-ICE 2000 universal in-circuit emulator. Available now in PDIP, SOIC, and SSOP packages, the PIC16F872 is \$3.34 each in 1,000-unit quantities.

Microchip Technology

Chandler, AZ
(480) 786-7200
www.microchip.com

MCU and DSP in one core

The TC10GP is a single-core computing engine that provides real-time microcontroller and DSP functionality, while using a single application development environment and system design tool chain. Based on the TriCore Unified Processing architecture, the TC10GP can serve as a stand-alone processor for board-level system designs, and as a development and

time-to-market platform for products that will ultimately incorporate more highly integrated system-on-a-chip devices based on the TriCore architecture. The chip comprises a TriCore 32-bit unified processor core running at 66MHz, 48K of on-chip memory, and a suite of on-chip peripherals for system connectivity, communications, programming, timing, and systems test. It achieves up to 100 MIPS of combined MCU and DSP throughput. Features include configurable cache and program memory, including on-chip SRAM; superscalar implementation with up to four operations per cycle; 16- and 32-bit instruction format; low interrupt latency and fast context switching; and several levels of power management. Volume production is scheduled for the second quarter of 2000. The chips are \$20 each in 1,000-unit quantities.

Infineon Technologies

San Jose, CA
(408) 501-6000
www.infineon.com

Errata

In the article "Overlaps Between Microcontrollers and DSPs" by Bill Giovino (January 2000, p. 20), a reference is made on p. 29 to the value of bit manipulation on "serial function registers." This should read "special function registers." On p. 34, the reference to Siemens Semiconductor should be to Infineon Technologies.

The price printed for the TimeWiz software from TimeSys Corp. (New Products, December 1999, p. 109) is not current. Pricing varies based on options. Please contact TimeSys directly for pricing information.

Embedded Systems Programming regrets the errors.

**DESIGN/DEVELOPMENT
ENGINEERS:**



OPPORTUNITIES IN:

Wireless Communications (Data, PCS, Cellular, Networks, Satcom),
Digital Imaging, Computers, Software, Semiconductors, Medical,
CATV, Defense, Process Control, Consumer Electronics

SKILLS IN ANY OF THE FOLLOWING:

Embedded SW, OOD/OOP, C, C++, WindowsNT/98, Solaris/UNIX,
JAVA, BIOS, VRTX, PSOS, DSP, MIPS, PCI, VME, Mixed Signal,
ASIC/FPGA, VHDL/Verilog, Device Drivers, System Architecture,
LAN/WAN, IP, Wireless Design, CDMA, GSM, Video Compression

National Engineering Search

is the leading search firm placing
Engineers nationwide. Contact us
for immediate access to today's
most exceptional engineering career
opportunities! Our clients range
from Blue Chips to today's newest
emerging technology companies.

800/248-7020

Fax: 800/838-8789

esp@nesnet.com

*See many of our current
opportunities on-line at:*

nesnet.com

**What are you worth?
See our Online Salary Calculator!**

scientific.com

**Software and Hardware
Professionals**

Nationwide opportunities available for:
Engineers/Designers

Quality Assurance/Testers/Verification

- Embedded SW
- RTOS
- ASIC/FPGA
- IC Design
- C/C++
- Firmware
- VLSI
- Java

Scientific Placement, Inc.

800-231-5920 800-757-9003 (Fax)

crs@scientific.com

**Free Resume Assistance
All fees are employer paid**

workingEngines[*inc]

Post - PC Era Talent Firm

→ your embedded career
options source

/*placing engineers across North America */

303.628.5560

www.workingEngines.com

{Complimentary resume & interview support.
All fees employer paid.}

Rent the
Hottest List
around!

Embedded Systems
PROGRAMMING

Over 50,000
Qualified Subscribers



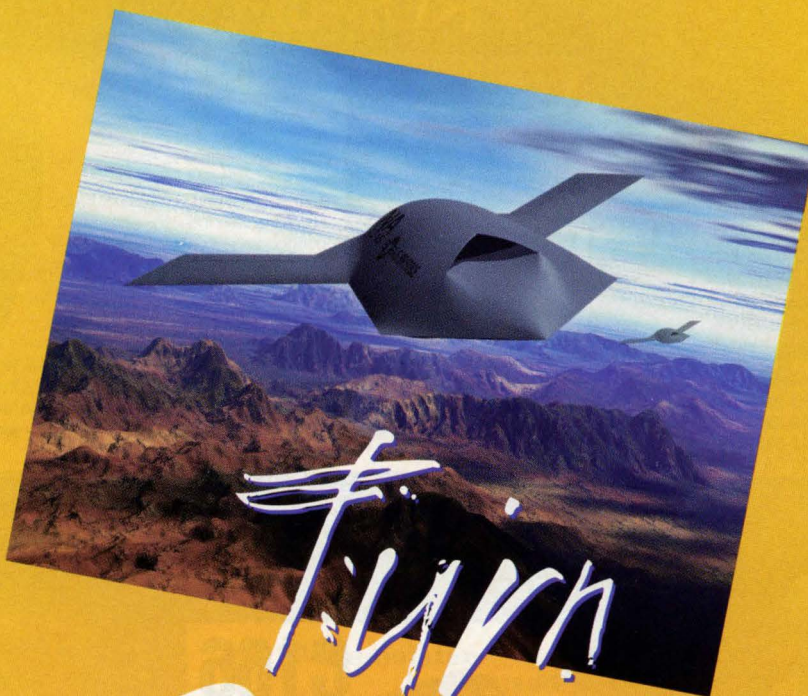
Call Rubin Response 847-619-9800



Rubin Response Management Services, Inc.

1111 Plaza Drive, Suite 800, Schaumburg, IL 60173
(847) 619-9800, Fax (847) 619-0149, kristenk@rubinresponse.com

request a quote on-line at www.rubinresponse.com



Turn
Science
Fiction
into Science
Non-Fiction.

www.boeing.com/employment



In-Circuit Emulators

• 8051 • 80186 • 80196
• DSP 320Cxxx • TriCore • PIC

- Real-time trace
- Serial, parallel, & LAN interface
- C/C++ level Chameleon Debugger
- Multi-core debugging

**FREE
2 WEEK
TRIAL**

www.signum.com

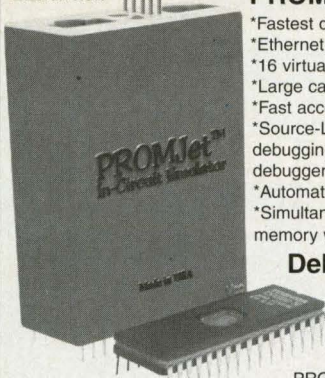
**SIGNUM
SYSTEMS**

800-838-8012
805-523-9774
Fax: 805-523-9776
11992 Challenger Ct. • Moorpark, CA 93021

PROMJet®

DebugJet®

2.2x1.7x0.7"



PROMJet

- *Fastest data transfer 1.2-4Mbit/S.
- *Ethernet or Parallel connection.
- *16 virtual COM channels.
- *Large capacity (up to 32Mb).
- *Fast access time (up to 25ns).
- *Source-Level and RTOS task-aware debugging with industry leading debuggers.
- *Automatic support for 2-5V targets.
- *Simultaneous access to emulation memory while target is running.

DebugJet

- *Supports standard JTAG protocol for testing and debugging.
- *Supports BDM protocol for CPU32 and Power PC debugging.
- *Optional TraceJet for hardware tracing and Breakpoint support.
- *Target network connectivity without ethernet on target by using PROMJet.



30-day money-back guarantee

EmuTec Inc. Tel: 425.267.9604 Fax: 425.267.9507
Web: www.emutech.com Email: emutech@emutech.com

Emulators that change
so you don't have to.



iSYSTEM™ in-circuit emulators and software preserve your investment in time and capital. Migrate to any of 300+ devices with a simple swap of the pod or a software setup. And retain the same user interface with winIDEA™, our integrated development environment. Adaptable, powerful, and easy to use—that's great value.

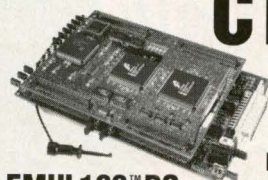
SUPPORT FOR
68HC05/08/11/12/16
68K, 683XX, MCORE,
MPC, 8051, 80X86,
C500, Z80/180, CR16,
PIC AND MANY MORE.

Call for your free demo CD
1 888 543-5300

iSYSTEM

usa@isystem.com
www.isystem.com

C166



EMUL166-PC

In-Circuit
Emulator

Real-Time Microprocessor Development Tools

Call (408) 866-1820 for a product brochure
and a FREE Demo Disk.
Information is also available via Fax, call our
24-hour Fax Center at (408) 378-2912.
Visit our web site- <http://www.nohau.com>

See EEM '97- pages D 1274-1282

**NOHAU
CORPORATION**

51 E. Campbell Avenue
Campbell, CA 95008-2053
Email: sales@nohau.com

In-Circuit Emulators

The Complete Package for Professional Embedded Systems Development Tools

**AVOCET
SYSTEMS®, INC.**

- C Cross Compiler
- Macro Assembler
- Simulator Debugger
- RTOS
- In-Circuit Emulator
- Universal Programmer
- Extensive Chip Support
- Quality Tech Support

(207) 236-9055 (800) 448-8500
P.O. Box 490, Rockport, Maine 04856
sales@avocetsystems.com
www.avocetsystems.com

THE JACKRABBIT™ FEATURE-PACKED CONTROLLER



\$49 qty. one

Gain high-performance control at minimal cost with Z-World's JackRabbit™ single-board computer.

- C-programmable
- 40+ I/O include digital I/O, RS-232/485 serial ports, A/D & D/A converters and high-voltage outputs
- Fast clock to 22MHz
- 6 timers & watchdog

DEV KIT ONLY \$139

Includes:

- JackRabbit™ single-board computer
- Dynamic C® development software and documentation on CD-ROM
- Prototyping board
- Power supply
- PC serial cable

ZWORLD

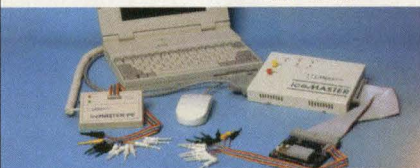
Order online @
www.zworld.com
or call toll free 888.362.3387

2900 Spafford Street Davis, CA 95616 USA Tel: 530.757.3737 Fax: 530.753.5141

IN-CIRCUIT EMULATORS

The Right Emulator at the Right Price!

8051 • 68HC05 • 68HC11 Analog Devices • Atmel • Dallas
COP8 • CR16 • DS80C320/520/530 Intel • Motorola • National • NEC • OKI
rx65K • 78K4 • SM6000/SM8500 Philips • Sharp • Siemens • SST
Temic • Winbond



- Windows-Based Interface
- Real-time/Non-intrusive
- Source-level debug
- Rentals available

**MetaLink®
Corporation**

Headquarters, Arizona, USA

Tel: 480.926.0797
Fax: 480.926.1198
email: sales@metalice.com
<http://www.metalice.com>

MetaLink - Europe GmbH

Tel: 49 (8091) 56960
Fax: 49 (8091) 2386
email: islinger@metalink.de
<http://www.metalink.de>

Fast. Reliable. Affordable.



NEEDHAM'S DEVICE PROGRAMMERS are the easiest and most cost-effective way to read, program and verify 2716 - 8 meg EPROMS. Support for Micros, Flash, EPROM, 16-bit, PLDs, Low Voltage and Mach (call for support list for specific models, or download demos from our BBS or web site). Easy to use menu driven software features on-line help, and a full-screen editor. Support for macros, read and save to disk, and split and set options.

- Free technical support • Free software upgrades
- 1 to 2 year warranty on all parts and labor
- 30-day money-back guarantee • Made in the U.S.A.
- All models include software, on-line help, cables, and power transformers (where applicable)

NEEDHAM ELECTRONICS, INC.

4630 Beloit Drive, #20, Sacramento, CA 95838
FAX (916) 924-8065 • BBS (916) 924-8094
(Mon. - Fri. 8 am - 5 pm, PST)
<http://www.needhams.com/>

(916) 924-8037

MasterCard VISA

CE

Debugger+Programmer

Complete System \$375

FREE

Demo/Prototype board

All tools **qualified** by
Scenix Semiconductor

SX-ISD

- Supports SX18/20/28/48/52 • In-circuit run-time debugging • Real-time code execution
- Source level debugging • Built-in programmer
- Real-time breakpoint • Conditional animation break • External break input • Frequency synthesizer • Selectable internal frequency
- External oscillator support • Software trace
- Unlimited watch variables • Parallel Port Interface • Runs under Win 95/98/NT
- Comes with SASM Assembler

Single, Gang Programmers and SMT adapters
are also available

Advanced TransData
CORPORATION

Dallas, Texas

Tel 972.980.2960

www.adv-transdata.com

PIC Real-time 'Emulator'

\$159



PIC-ICD

PIC-ICD

= **Debugger + Programmer + DemoBoard**

- Designed for 16F87X • Can also support most 16C6X/7X • In-circuit run-time debugging
- Real-time code execution • 2.5-6V operating voltage • Source level debugging • Built-in programmer • Real-time breakpoint
- Conditional animation break • 2 External break inputs • Selectable internal frequency
- Software trace • Unlimited watch variables
- Parallel Port Interface • Runs under PICICD IDE (Win95/98/NT) or MPLAB (Win95/98)

Advanced TransData
CORPORATION

Dallas, Texas

Tel 972.980.2960

www.adv-transdata.com

PIC16/17 Emulator

New Low Prices

Complete system
starts at **\$599**



RICE17A

- For PIC12/16/17 • 3.5-volt emulation • 64K program memory • 32K real-time trace
- 12-clip external probe • Source level debugging • External break input • Trigger and break output • Real-time breakpoint
- Unlimited software break and trigger points
- Selectable internal frequency • Unlimited watch variables • Parallel Port Interface • Runs under Win95/98/NT

Probes for **16F87x** and **16C77x** by Jan 2000
With data break support

Advanced TransData
CORPORATION

Dallas, Texas

Tel 972.980.2960

www.adv-transdata.com

Vesta Technology, Inc.
Single Board Computers for the 21st Century

SBC2000-332

Vesta **BASIC**
or **C** Software
32 bit 68332
Low Power
2 x RS-232
Watchdog
Real Time Clock
EPROM
LCD/Keypad
1 Meg FLASH
1 Meg ROM
1 Meg RAM
\$299 @ 1
\$188 @ 100

(303) 422-8088

www.sbc2000.com

**"The best emulator
I ever used!"**

8051



- more than 500 derivatives supported
- small emulation probes
- real-time access to internal bus
- can trigger on internal bus events
- cascading triggers
- trace with timestamps
- dual-ported emulation memory
- external trace and triggers
- excellent HLL support
- code coverage
- performance analysis

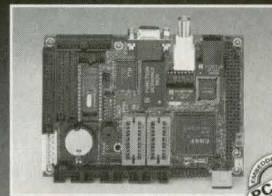
hitex

DEVELOPMENT TOOLS

1-800-45-hitex

www.hitex.com

386SX40 PC104 Embedded Controller



ALI M6117D Intel compatible 386SX40Mhz

- 4MB System DRAM on board
- HMC HM86508 VGA Chip w/1MB DRAM
- Support DiskOnChip up to 144MB
- REALTEK 8019AS Ethernet on board
- 4S/1P/IDE/KB/FDD/DIO on board
- 7 Level Watch Dog Timer
- Support DOS, Window 3.X, Linux, XvWork, and QNX
- 145mm x 102mm (5 3/4"x4"), 2 side PCB

Other PC104 relative products available:
486SX, AC/DC, Digital I/O, Ethernet Link, 4 Port RS232

NUCLEUS Electronic Corp. **800-683-7335**

Tel: (909) 468-5700

Fax: (909) 468-5704

<http://www.nucleus1.com>

Email: info@nucleus1.com

Serial in, graphics out.
Almost too easy.



G12032 Serial Graphics Display
120x32-pixel LCD w/backlight

Jump-start your design project with our easy-to-use serial graphics LCDs. Works like a tiny terminal at 2400 or 9600 bps. Stores custom fonts, bitmap screens in EEPROM. Order today, show a product with a graphics display tomorrow!

www.seetron.com

Scott Edwards Electronics, Inc.

phone: 520-459-4802 fax: 520-459-0623

Windows[®] CE
Windows Is A Registered Trade Mark Of Microsoft Corp.
Embedded Computer

Visual Basic or Visual C/C++

Small Powerful & Easy To Program.

Plug the CE-Minus-SC400 (486 SBC) into your next custom design or into our CE-Plus and LCD-Plus I/O expansion options. The CE-Minus is easy to program using our custom ported Windows-CE & Tools.

www.RLC.com
RLC Enterprises, Inc.
Toll Free 1-888-RLC-TECH

CE-Minus
Single Board Computer

\$199 Qty 100



2.5" x 4.5"

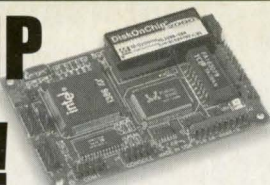
1/4 VGA LCD
320 x 240

Monochrome Display



Touch Screen Standard

JUMP on the NET!



The μFlashTCP gives you 10BASE-T Ethernet connectivity, a full-function TCP/IP stack and 2 serial ports in a package 30% smaller than PC/104 solutions.

Field-proven TCP/IP stack, DOS and PC-compatible BIOS make development quick and easy.

Prices start at \$169 qty 100. Development kits are available.

Call 530-297-6073 Fax 530-297-6074

Check our web site for the latest updates

www.jkmicro.com/uflash

JK microsystems

From the Author
of WATTCP

eRTOS

DOS
Realtime
Kernel with
TCP/IP Support

- Preemptive & Cooperative threads
- Critical Section Protection
- Interthread Messaging
- Complete Re-entrant TCP/IP
- Web, CGI, FTP, Email, Telnet
- Web Graphics
- Interrupt-driven Serial Support

www.ertos.com

Call 530-297-6073 Fax 530-297-6074

JK microsystems

UniSTAC™ for (SA-1110, SA-1100)
Strong ARM designs



In-Circuit Emulator Features:

- 64K frames Real-time Bus Trace
- 8 Mbytes emulation memory
- 1 hardware breakpoints
- Unlimited software breakpoints
- Program performance analysis and code coverage
- Test target included

UniSTAC is a full-featured, high-end development system for Strong ARM™ SA-1110 designs.

Powerful In-circuit Emulator featuring: 8 Mbytes emulation memory, non-intrusive Real-time trace capture and display, support for hardware breakpoints.

Sophia Interface with target:

- Sophia original connector-without removing target CPU
- Adapters also available for BGA256

Advanced GUI source level debugger 'Watchpoint'

Sophia's powerful high-level language debugger hosted on Windows® 95/98, WindowsNT™

Also Supported:

Power PC, ARM7/9186/386/486, Pentium, 680x0, SuperH, V800/850E, R3000/4000, M16C, M32R

Sophia systems™

408-467-9911

www.sophia.com

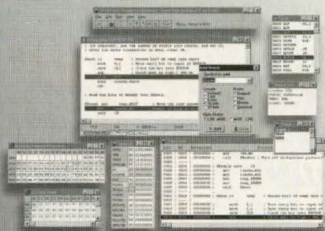
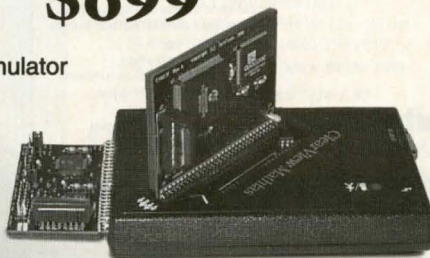
Embedded Systems Development Tools



Complete PICmicro® Development System

Get the TOTAL Package for only
\$699

PICmicro® In-Circuit Emulator
PICmicro® Assembler
PICmicro® Debugger
Windows® IDE



ClearView™ Mathias is a full-featured In-Circuit Emulator with a highly productive Development and Debugging environment for the PICmicro.

Fully emulates the selected PICmicro, including program memory, register memory, EEPROM, I/O activity, SLEEP mode and all peripherals.

Intuitive, Easy to Learn full-featured environment with integrated ClearView Debugger.

TDE provides integrated source-level debugging for ALL popular PIC Compilers and Assemblers.

See why Engineers choose TechTools

Visit our website and request your FREE CDROM!

www.tech-tools.com

CONTACTS: (972) 272-9392 Fax: (972) 494-5814 Email: sales@tech-tools.com

Copyright © 1999, TechTools, P.O. Box 462101, Garland, Texas 75046-2101 • ClearView, CVASM16, PICwriter, PICstation, the "Wizard" symbol and TechTools are trademarks of TechTools, P.O. Box 462101, Garland, Texas 75046. • PICmicro is a registered trademark of Microchip Technology, Inc. • All other trademarks are trademarks or registered trademarks of their respective company.

SB-56K Multi-DSP Emulator



Support for the Motorola DSPs:
DSP560xx, DSP563xx, DSP566xx, DSP568xx

SB-56K supports any combination and any count (up to 255) of the devices from the above families. With its accurate counter allowing to measure code execution (benchmarking), small size (1"x2.5"x4"), high speed RS-232 interface, the SB-56K can provide independent support for multiple devices with option to access each device on the target board from different workstations connected through LAN, WAN or Internet.



1700 Alma Dr., #495, Plano, TX 75075
Tel.: (972) 578-1121 / Fax: (972) 578-1086
info@domaintec.com
www.domaintec.com

PromICE
with Trace

The Leader in Memory Emulation

- Trace to pinpoint startup problems and isolate real-time bugs.
- Code Coverage to verify execution and speed up QA.
- Ultra-fast downloads via Ethernet, parallel and serial ports for Unix, Windows 95/NT and DOS.



Grammar Engine Inc.

Call Toll Free:

1-800-776-6423

www.gei.com

NOW!

Optimised
ANSI C
on the
Microchip
PIC.

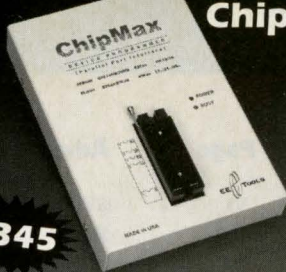
the
world's
first

pic
ANSI C
COMPILER

CALL NOW
1-800-735-5715

Fax 1-407-722-2902
www.htsoft.com/pic... sales@htsoft.com

HI-TECH
SOFTWARE



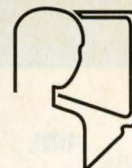
ChipMax

\$345

- Programs over 1200 devices (E)PROM, Flash, Serial, PALCE, GAL, 875x/895x, 93Cxx, 17xx, PIC16xx)
- Fast parallel link to any PC, even laptops
- 40-pin universal pin driver and current limit
- On-board processor and built-in power supply
- Unbeatable programming speed
- Checks for incorrect device insertion
- Automatic EPROM ID search
- Supports WIN95/98/NT
- NO fans & modules are required in circuit
- Made in USA, Lifetime free software
- Visit web site for demo software



Sunnyvale, California, USA
Tel: 408-734-8184
Fax: 408-734-8185
www.eetools.com



**Blunk
Microsystems**

\$5K RTOS

\$5K LAPB

\$5K TCP/IP

NEW! **\$5K** Flash File System

☒ Royalty-free

☒ Source code included

☒ **CodeWarrior™** CPU32
and MPC860 Integration

(408) 323-1758

www.blunkmicro.com

68HC12

Real-Time Debugging



- DG128, DA128, D60, BC32, B32, A4 etc...
- Flash and EEPROM programming
- Page mode debugging
- Hardware breakpoints
- Hot plug
- Cosmic, Hiware, IAR, Introl support

Noral Micrologics, Inc.
**The Debugging
Company**

Call toll free:
888-88-DEBUG
for free
evaluation



Tel: (508) 647-0103
Fax: (508) 653-1828
harrye@tiac.net
http://www.noral.com



- Real-time debugging
- 5307, 5206e, 5206 etc
- Real-time trace
- Multi-level breakpoints
- 90MHz
- 2.7 to 5.5V
- Diab, GNU, Green Hills, Microtec, Tasking, Crossware support

Noral Micrologics, Inc.
**The Debugging
Company**

Call toll free:
888-88-DEBUG
for free evaluation

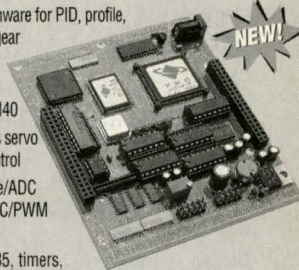
Tel: (508) 647-0103
Fax: (508) 653-1828
harrye@tiac.net
http://www.noral.com



MotionC™
2140

**C/C++ programmable
Low Cost, Standalone
DSP Motion Controller**

- Built-in firmware for PID, profile, electronic gear
- 4.7" x 3.8", 386EX/186 PMD MC2140
- 2- or 4-axis servo motion control
- Quadrature/ADC inputs, DAC/PWM outputs
- RS-232/485, timers, watchdog, TTL I/O, ADC



We have 30+ Low Cost Controllers with ADC, DAC, solenoid drivers, relays, PCMCIA, LCD, DSP motion control, 10 UARTs, 300 I/Os. Custom board design. Save time and money!



1724 Picasso Ave., Suite A
Davis, CA 95616 USA
Tel: 530-758-0180 • Fax: 530-758-0181
http://www.tern.com
tern@netcom.com

Phar Lap Software, Inc.

Tel: (617) 661-1510
Fax: (617) 876-2972
www.pharlap.com
info@pharlap.com



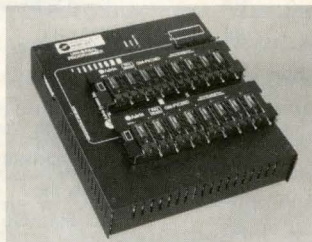
**Win the "Time to Market" Race
and put yourself ahead of the competition**

TNT Embedded ToolSuite
**Realtime Operating System &
Embedded Development Tools**

- RTOS supports Win32 API
- MS Visual C++ 6.0 support
- MS Developer Studio support and Embedded Studio Express add-in
- Realtime GUI
- TCP/IP – Winsock API featuring SNMP, WinInet and Microweb Server.



Advin



Production and Engineering Programmers
**Extensive support for
Microchip PICs and others,**
Wide variety of package types supported.

*Thousands of happy customers worldwide
are living proof of our product quality.*

**Please contact us and find out why
Advin is your best choice.**

– Accept trade-ins of out-dated Data I/O models –
www.advin.com
1-888-GO-ADVİN 408-243-7000

GALEP-III
Pocket Multiprogrammer

**This size
fits all!**



- Programs 8-bit and 16-bit EPROMs, EEPROMs, Zero Power RAMs, Flash, serial EEPROMs • GAL, PALCE, ATF • 87xxx, 89xxx, PIC12/16/17Cxx • All DIL devices without adaptor • **Lightning fast** parallel data transfer (e.g. 27C512 read/compare 2 sec!) • Power supply independent due to **rechargeable battery** • Uses PC printer port • Hex, JEDEC, and binary file formats • Hex and fusemap buffer editor • Split & shuffle for 8-bit, 16-bit and 32-bit targets • Runs under Win3.1, 95, 98 • 'Remote control' by DDE scripts • Designed for the future due to flexible pin driver technology - new devices will be added every month • Device list, demo software and **lifetime free updates** from our website www.conitec.com!

GALEP-III Set with cable, battery, recharger... \$333.00

PLCC Adaptor for 8-bit EPROMs / 16-bit EPROMs / GALs each \$149.00

CONITEC 1951 4th Ave, Ste 301 • San Diego, CA 92101
Tel: (619) 702-4420 • http://www.conitec.com

Advertiser	URL	Page	Advertiser	URL	Page
Accelerated Technology	www.atinucleus.com	93	iSYSTEM GmbH	www.isystem.com	108
Advanced Transdata Corp.	www.adv-transdata.com	109	JK Microsystems	www.jkmicro.com/uflash	110
Advantech Technologies Inc.	www.advantec.com/epc	79	JK Microsystems	www.jkmicro.com	110
Advin Systems Inc.	www.advin.com	111	KADAK Products Ltd.	www.kadak.com	41
Agilent Technologies	www.agilent.com	21	Keil Software	www.keil.com	11
Agilent Technologies	www.agilent.com	102	Keil Software Inc.	www.keil.com	101
American Arium	www.arium.com	C3	KW Software	www.kw-softwareusa.com	49
Archimedes Software	www.archimedessoftware.com	104	Lauterbach	www.lauterbach.com	10
Arcom Control Systems	www.arcomcontrols.com	88	Lauterbach	www.lauterbach.com	78
ARTISAN Software Tools	www.artisansw.com	29	Lynx Real-Time Systems	www.lynx.com	69
Ashling	www.ashling.com	12	Lynx Real-Time Systems	www.lynx.com	15
Avocet Systems Inc.	www.avocetsystems.com	C4	MetaLink Corp.	www.metaice.com	108
Avocet Systems Inc.	www.avocetsystems.com	108	Micro Computer Control	www.mcc-us.com	11
Berkeley Software Design	www.bsdi.com	71	Micro Digital Inc.	www.smxinfo.com	99
Blunk Microsystems	www.blunkmicro.com	111	Micro/sys Inc.	www.embeddedsys.com	98
Boeing Recruitment	www.boeing.com/employment	107	Microsoft Inc.	www.microsoft.com/embedded	35
Byte Craft Ltd.	www.bytecraft.com	59	Microtek International Inc.	www.microtekintl.com	4
CAD-UL	www.cadul.com	65	National Engineer Search	www.nesnet.com	106
ChipTools	www.chiptools.com	12	National Semiconductor	www.national.com	75
CMX Systems Inc.	www.cmx.com	42	Needham's Electronics	www.needhams.com	108
CMX Systems Inc.	www.cmx.com	12	Nohau Corp.	www.nohau.com	11
Cogent Computer Systems Inc.	www.cogcomp.com	98	Nohau Corp.	www.nohau.com	C2
CONITEC Datasystems Inc.	www.conitec.com	111	Nohau Corp.	www.nohau.com	108
COSMIC Software	www.cosmic-software.com	70	Noral Micrologics	www.noral.com	111
DIAB-SDS, Inc.	www.diabsds.com	77	Noral Micrologics	www.noral.com	111
Dinkumware, Ltd.	www.dinkumware.com	98	Nucleus Electronics Corp.	www.nucleusl.com	109
Domain Technologies	www.domaintec.com	110	Pacific Softworks	www.pacificsw.com	73
EBS	www.etcbin.com	98	Paradigm Systems	www.devtools.com	86
EBSnet	www.etcbin.com	57	Parasoft Corp.	www.parasoft.com	32
Electronic Engineering Tools	www.eetools.com	111	Phar Lap Software	www.pharlap.com	111
Electronic Warfare Associates	www.ewa.com	31	Phytec	www.phytec.com	12
EMAC Inc.	www.emacinc.com	76	QNX	www.qnx.com	91
Embedded University	www.embeddedU.com	94	Rabbit Semiconductor	www.rabbitsemiconductor.com	43
Embedded University	www.embeddedU.com	9	Radisys	www.radisys.com/SS7	23
Emulation Solutions	www.adapters.com	12	RLC Enterprises	www.rlc.com	109
EmuTec	www.emutec.com	108	Scientific Placement Inc.	www.scientific.com	106
Enea OSE Systems	www.enea.com	85	Scott Edwards Electronics Inc.	www.seetron.com	109
Esmertec, Inc.	www.esmertec.com	47	Signum Systems	www.signum.com	108
EST Corp.	www.estc.com	1	Sophia Systems	www.sophia.com	110
Express Logic	www.expresslogic.com	87	Swell Software	www.swellsoftware.com	36
General Software	www.gensw.com	84	TASKING	www.tasking.com	22
Grammar Engine Inc.	www.gei.com	110	TechTools	www.tech-tools.com	110
Green Hills Software Inc.	www.ghs.com	11	Tern Inc.	www.tern.com	111
Green Hills Software Inc.	www.ghs.com	6	The Math Works	www.mathworks.com	45
HI-TECH Software	www.htsoft.com/pic	111	TqComponents	www.tqc.de	10
HI-TECH Software	www.htsoft.com	89	Treck Inc.	www.treck.com	53
HI-TECH Software	www.htsoft.com	12	U.S. Software Corp.	www.ussw.com	54
Hitex Development Tools	www.hitex.com	83	U.S. Software Corp.	www.ussw.com	12
Hitex Development Tools	www.hitex.com	109	Vesta Technology	www.sbc2000.com	109
Hitex Development Tools	www.hitex.com	10	Wind River Systems	www.wrs.com	26,27
Hiware	www.hiware.com	72	Working Engine Inc.	www.workingEngines.com	106
IAR Systems	www.iar.com	38	WSI	www.waferscale.com	10
Infineon Technologies	www.spacetools.com	9	WSI	www.waferscale.com	96
Insignia Solutions	www.insignia.com	60	XILINX, Inc.	www.xilinx.com	51
Integrated Systems Inc.	www.pOSEK.com	11	Z World	www.zworld.com	108
InterNiche Technologies Inc.	www.iniche.com	62			

This page is provided as a service to readers. The publisher does not assume any liability for errors.



Jack G. Ganssle

Reality Bites

The real world is a nasty place. Dust mites, politicians, and noisy analog signals are all symptoms of the grimy side of reality. Only the foolish assume that the world—and signals—are pristine Gardens of Eden.

After too many years building embedded systems, I'm used to dealing with imperfect systems and signals. I was brought up short, though, when recently dealing with a number of folks who assumed that ones are ones, zeros are zeros, and all data is perfect all of the time.

This past fall I co-taught an electrical engineering lab course at the University of Maryland. Friends out there in reader-land will chortle at this, given my, ah, less than stellar academic career. The irony of teaching a senior level class at the very same institution that, 25 years ago, pretty much tossed me out was just too delicious to pass up.

So after enduring incredible bureaucratic approval hassles, I found myself with a faculty ID card and a room full of eager youngsters looking for pearls of wisdom. Precious few pearls appeared, but I sure learned a lot. With just a few exceptions these nascent engineers had no knowledge of the grimy nature of the world they're about to embark on, nor did they have a "feel" for the nature of the electronics.

EEs learn plenty of theory in their university years, theory that's essential to developing an understanding of how things work. But a huge gap exists between knowing the formulas and applying them in practical circuit design.

Some of the best engineers I've met started their careers as technicians,

perhaps delaying, or in some cases, completely deferring a college experience. Techs, much more so than many engineers, build things, get their hands dirty, replace components, and troubleshoot circuits. Sometimes I think we learn best when we learn with our hands, by doing things, manipulating the environment using tools and practical skills.

Consequently, technicians—or at

As an instructor I found the students offered an astonishing flow of often brilliant ideas for firmware designs. Some seemed reasonable, but just felt wrong. My instinctive gut reaction preceded a more detailed analysis, yet all too often was just as correct as the rational explanation. Feel, whether for electronics or firmware, comes from long bitter experience. It's a critical and effective tool.

The difference between theory and practice is much like the difference between logic analyzers and oscilloscopes.

least engineering techs—gain an intrinsic feel for the properties of electrical circuits and components. It's something that can't be taught in a classroom, but that comes only experientially. Though the theory does predict these characteristics, there's simply so much theory that most folks have a hard time translating four years' worth of accumulated formulas into practical cause and effect. Ask a young engineer what happens when you insert a tantalum capacitor backwards, and you'll get a shrug, a "perhaps it won't work?" The tech and the experienced engineer know that if you don't take cover capacitor bits and pieces will bounce off your glasses. A newbie may panic when touching a processor and feeling some heat; a tech's calibrated-by-experience fingers assess the temperature and he or she immediately decides if the part is operating in a normal fashion.

Ones are not ones

Most young engineers are under the mistaken impression that a "one" or a "zero" is a real thing. They look at the scope and see a voltage greater than nothing and grandly pronounce the signal is "high," often without even knowing the instrument's volts per division setting.

There ain't no such thing as a "one." The concept is an abstraction, a mere shorthand to ease our description of electron flow in a very real electronic circuit. No one who uses C thinks of it as the computer's native language; it's a useful abstraction from the muddy waters of assembly language. The dark side of abstractions lies in accepting the easy environment they create: wise C programmers know that it makes sense to look at the generated assembly code from time to

time, and wise developers understand that a one is only a one if it meets the specs of the components in use.

Ones and zeros represent voltage levels, which in turn come from electron flow. Fact is, electrons are difficult beasts that have mass, spin, and charge. These physical realities intrude on the seeming perfection of the computer domain. Shoot a one down a wire and, because of real physical properties, the gate at the receiving end may see a zero. Or a bouncy one. Or even a succession of pulses. In degenerative cases, that imagined perfect one might induce SCR latchup in the receiver, literally causing the gate to explode.

Thus an understanding of electromagnetics theory helps us predict what happens in the environment of a real PC board. It also suggests that an inexperienced theoretician may never imagine that unless Maxwell's tremendously difficult equations are applied, a one can turn into a "killer one" that destroys ICs. Only painful experience—the developed "feel" for these problems—teaches us just the necessity for including Maxwell in our estimations of the behavior of even a simple logic circuit.

Of course, another difference between a theoretician and a practicing engineer is learning how to shortcut the hard math. I prefer what might be called Maxwell's rules of thumb, simplified ways of applying electromagnetics to get the job done. For the best book on that subject see *High Speed Digital Design (a Handbook of Black Magic)* by Howard Johnson and Martin Graham (Prentice Hall, 1993).

One of my challenges this semester was trying to convince students to use their scopes instead of the logic analyzer. Not that there's anything wrong with analyzers; these wonderful tools fit a wide range of problems. But they often hide the electronic truths behind the one/zero abstraction.

Don't forget that every logic analyzer converts an analog voltage sensed in your circuit to a one or zero based on rules built into the instrument. On some these rules are programmable;

others might assume that TTL voltage levels define the correct interpretation of "oneness." That assumption is false.

A signal interpreted as a zero by the analyzer may indeed be—for the logic family you're using—a real zero. Or it might be a zero struggling to be a one, perhaps partially clamped by bus contention. Maybe it's a one that can't quite make it high due to excessive loading or a hundred other reasons. The logic analyzer does what it's supposed to do, converting real voltages into logic levels based on these rules, completely hiding the electronic facts of life from the user.

Signals on even the most benign and TTL-compatible of microprocessors may define logic levels differently on each pin. An awful lot of micros that happily coexist with TTL's 2V ones require four or more volts on the reset and/or clock inputs.

The oscilloscope is the only tool that cuts through the one/zero abstraction layer to show the ugly realities of your signals. A scope might not be appropriate for working with buses, but it's the best tool for working out circuit problems on individual signals, especially when they get translated by transistors or other devices.

A scope is useful, though, only if you steadfastly refuse to fall into the one/zero mindset. Using TTL logic, a one might be 2V. That suggests to me that it's critical to set up the instrument in a way that causes it to clearly show the difference between 2V ones and one wannabes. That is, if the vertical channel is set to 5V/division, you simply cannot resolve the difference between a decent "one" and a signal that has problems.

Worse, most scopes are a sea of knobs. Are you really sure the unit is set to, say, 1V/division? Remember that the probe itself often attenuates the signal by an order of magnitude. The only sure way to check the scope's vertical setting is to put the probe on the calibrator output, generally a square wave supplied to a front panel post. Typically this is a half-volt waveform, adequate to check your gain settings. I have an HP mixed signal scope (a scope/logic ana-

lyzer hermaphrodite) that provides a 5V calibrator output, a much more useful level for digital debugging.

The ancient analog scopes in the lab were a source of never-ending complaints. For the first few weeks I silently agreed, thinking that in this day and age we should have reasonable equipment. Over time I found that these oldies forced them to think more, to learn to fully use the triggering subsystem, and to interpret the data more carefully. Yes, a modern digital oscilloscope might perform one-touch measurements, but all instruments lie. Without a deep understanding of what the device does, we may miss the fact that a knob is set incorrectly, or the pattern we see is really an aliased waveform.

Resistor realities

In fairness to my class, many of the students were in the computer engineering curriculum, and not electrical engineering. It seems computer engineering is largely focused on digital electronics. Part of me acknowledges that the fields are so big some sort of specialization is needed; part of me shivers when thinking of this. Digital is, after all, just a special case of analog. Just as EEs learn physics to have a basis for understanding what goes on under the hood of their circuits, so I think CEs need a deep electronics basis for comprehending their computer creations.

Nothing made this so apparent as our struggles with resistor selection. Everyone knew that resistors come in different flavors, though few thought beyond differences in ohms.

One of the projects interfaced a microprocessor to a variety of 12V devices. A simple emitter-follower transistor circuit, with a 51-ohm resistor between the transistor and ground, amplified the "pure digital" outputs. Twelve volts and 51 ohms. Even without doing the math my "feel" says this sounds like a prescription for heat. Apply I^2R and, indeed, the poor resistor is dissipating about 2w.

The smell of burning 1/4-watt resis-

tors filled the room. The more experienced students recognized the unique smell of a hot resistor and shut their boards down. Others, perplexed, reacted more slowly as they tried to understand what was going on. "Troubleshooting by sense of smell" is not in the curriculum at this school, yet it's an incredibly useful skill.

Over the years I've found that so many recent graduates are similarly unaware of power dissipation issues that I've developed a standard teaching aid: connect a power supply to a 10-ohm, 1/4-watt resistor. Slowly crank up the voltage. The new EEs are always interested to see that as voltage goes up, so does the amperage. This confirms their knowledge that Ohm's Law is at play. Then, as we continue to increase the voltage, I have them touch the part. It's hot! Why? With some prompting they remember power dissipation. The voltage continues to go up, and they notice an odd smell. Soon the resistor is dumping smoke—a lot of smoke—until it bursts into flames.

This multi-sensory exploration of I^2R leads to an explanation of sizing resistors for watts as well as for ohms, something few EEs learn in school. The world is a complex place. Selecting resistors based on only their resistance may satisfy the equations governing the circuit's basic behavior, but may lead to designs doomed to failure.

We burned up an astonishing number of LEDs and optocouplers when students connected these parts directly between Vcc and ground. I suppose they viewed these components as light bulbs that just need "power," whatever that means. I'd suggest putting a resistor in series, and found folks using values ranging from 100 ohms to 10,000 ohms. It simply never occurred to many of them that the resistance mattered! $E=IR$ is burned into their brains by basic circuit design classes, but without context, without a deeply rooted sense that lots of current leads to destroyed parts.

One student asked about the gold color band, which indicates these particular parts have a 5% tolerance. He

seemed incredulous as I explained that there's an error associated with all of the parameters of real electronic components. "Some capacitors have a 50% tolerance or even worse," I told him. You could almost see him reevaluating his career choice as he considered the difficulty of designing with parts whose values are indeed a bit fuzzy.

Happily, we digital folks do have rather forgiving circuits, which usually tolerate quite a range of values. It's one of many reasons I abhor analog design. But we clearly must always understand that everything is shrouded in error bands; perfection just does not exist.

Debugging

One of our most creative endeavors is debugging our hardware and firmware. The red LED doesn't come on. Has the circuit failed? Is it designed incorrectly? What's the code doing? Debugging is creative because a relatively simple bug opens an entire world of possibilities. We're then tasked with searching through this universe of failure modes to isolate the one lousy bit that's wrong. Searching for the needle in the haystack looks like a lark by comparison.

And yet practicing developers become adept at chasing down defects. Some might challenge us for hours or even weeks, but most fall to our debugging onslaught in minutes. One satisfying part of teaching this course was watching the students' debug skills grow over the weeks and months.

At first, many were almost paralyzed by each bug. "What do I do? Where do I start?" They almost universally tried to out-think the code, running through listings. My mantra became "use your tools!"

Hiware had generously donated a set of cross development tools for the 68HC12 we used. Their powerful debugger, coupled to a BDM driving the target hardware, gave great insight into the system—when we used it. Early in the semester I saw student after student staring in desperation at the screen,

wondering how that function could return such bad data. Why were they so reluctant to use the debugger, which was already loaded and running (it was the vehicle for downloading code)? Perhaps the "why" is too philosophical; certainly, it's a question I never answered.

But the message I tried to give them was to get data. Use the tools to see what's going on. In this column over the last two months I've described embedded development in the bad old days when the tools were so much cruder. Now we're blessed with wonderful debuggers that show structure elements, interpret internal peripheral registers, and allow almost instantaneous edit/compile/download cycles. When I see a student or a practicing engineer who's reluctant to make the maximum use of the tools, employing every resource at hand, from breakpoints to logic analyzers to real-time trace, I wonder why that person is working so hard. The fundamental challenge of debugging an embedded system is finding a way to look inside of what is, by definition, a closed system.

Filling the gap

This isn't a rant against the students in this class. On the contrary, I was awed by the general intelligence of the group, which ranged from quite bright to brilliant. As their skills grew, they wrote code and found problems at a pace that left me dizzy.

What struck me was the huge gap between a grasp of theory and its practical application. Clearly, developing embedded systems is a funny endeavor, perhaps a bit like medicine. The practitioners acquire great knowledge, which experience tempers into a gut feel for how things should work. **esp**

Jack G. Ganssle is a lecturer and consultant on embedded development issues. He conducts seminars on embedded systems and helps companies with their embedded challenges. He founded two companies specializing in embedded systems. Contact him at jack@ganssle.com.

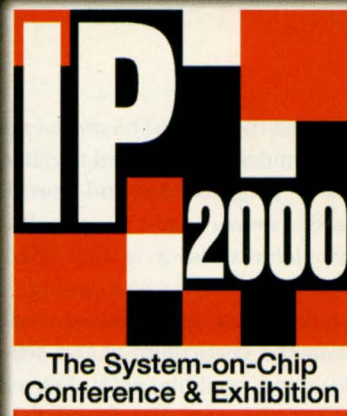


March 20-22, 2000, Santa Clara Convention Center

IP2000 is organized by

ElectronicsTimes

EETIMES



IP 2000 aims to provide a forum for the latest developments in design reuse and system-on-chip design. The main focus will be on the reality of intellectual property and indeed how design reuse will provide the urgently needed added-value product for the semiconductor industry.

A business forum plus a two-day conference and exhibition will address both the business and technology issues surrounding the use and reuse of semiconductor intellectual property and SoCs. As design reuse is seen as essential for an ever-increasing number of projects, the conference is expected to include a set of tutorials, in addition to presentations, on the latest developments in design with IP.

IP 2000 is the forum for discussion and debate of issues critical to the adoption of design reuse methodology.

Exhibiting companies:

(Correct on 12/10/99)

ARC Cores
Artisan Components
CoWare
DualSoft
Escalade
Hantro Products Oy
Improv Systems Inc.
Integrated Silicon Systems
Mentor Graphics
MoSys
NurLogic Design
PALMCHIP
Pivotal Technologies
Qualis Design
RocketChips
SICAN Microelectronics
Simutech
Sonics Inc.
Stellar Semiconductors
Summit Design
Synopsys
Synchronicity
Tensilica
The Alba Centre
UMC Group
VAutomation, Inc.
VCX
Virage Logic
VSI Alliance
Xentec Inc.

For further information on attending the conference, exhibiting or visiting the exhibition area, please visit the web site at

www.ip2000.com

IP2000 is sponsored by:





P.J. Plauger

Finding C Level

There's something about the entry into a new millennium that sets you to thinking back. (And please don't send me letters and e-mail telling me that the new millennium has yet to begin. When the odometer turns over at 100,000 miles, you've long since forgotten that the car had 2.7 miles on it when you got it new from the dealer.) I do that more and more these days anyway, given the stage of life I've now reached. Isaac Asimov called his mid-50s a state of "advanced youth," but I'm less sanguine. I feel more like a football coach in the fourth quarter of his most important game, with the clock running, with a moderately injured bench, still trying to figure out how to advance the ball for just a few more drives.

But I digress. It's the past I was thinking about. Nearly 30 years ago, in fact. For that was when the programming language C was invented, alongside the operating system Unix. I was fortunate enough to have a ringside seat to those early developments, at Bell Labs Murray Hill. I also had enough sense to know a good thing when I saw it. Since my earliest days as a programmer, I had majored in Fortran, with a strong minor in assembly language. I preferred writing in a high-level language, but never hesitated to drop down to assembly language if I needed the power or performance.

I quickly recognized C as a language that was better structured than Fortran, yet offered power and performance comparable to assembly language, at least for the vast majority of coding tasks I encountered. Those

coding tasks were pretty wide ranging, so I felt safe in concluding that C could, and should, replace assembly language almost all the time. And over the next couple of decades, I found that conclusion strongly reinforced in practice.

I wrote two or three C compilers, in C of course, as well as the accompanying library and all the run-time code needed to support C on a bare

though not necessarily thanks to my efforts. They have even penetrated deeply into the conservative bastions of embedded systems programming, always one of the last holdouts to change.

So why this insistence on moving up to C? It's not a magic bullet, to be sure. A good programmer can write good code in the worst of programming languages, and a bad programmer can

Friends don't let friends write assembly language—especially if a sensible choice is available.

machine. I targeted those compilers to a half dozen different architectures and several dozen different operating systems, as well as free-standing environments. In each case, I found it necessary to write no more than 500 to 1,000 lines of assembly language unique to each machine. I wrote an additional 500 to 1,000 lines of machine-specific or system-specific C code for each run-time environment. The vast bulk of the code was portable C. I also wrote a few operating systems, most small, but one fully Unix compatible in its day, with much the same experience.

Along the way, one of my self-appointed chores was to convince other programmers of the wisdom of converting from assembly language to C. Sometimes I was successful, sometimes not. But over time I acquired quite a few allies. C and other high-level languages are widely used today,

write crud in the best of them. We all know that design techniques are more important than the choice of language, that test methods have more impact on initial reliability, and that good coding hygiene has more influence on long-term maintainability. It would seem that the choice of programming language is almost incidental.

Well, it never really is. For any given project, you can probably choose among several candidate languages with the assurance that the particular choice is not all that important. But heaven help you if you get a serious mismatch between project size and language. Imagine programming a 16K controller for a coffee pot in Ada, or an air-traffic control system in Visual BASIC. You can probably do it, but it would be wrong.

The mismatch I'm addressing here occurs when programmers accus-

tomed to writing everything in assembly language work their way up to projects that are really too large for their accustomed techniques. It's not that they don't succeed—often they do, and with reasonable results. It's what they're missing out on by writing in a higher-level language if the project can sustain the small added overheads in code space and execution time. What you win in return is much shorter time to market and much greater code reuse.

By code reuse, I don't mean that you'll have a coffee-pot controller you can sell to others. Nor do I mean to suggest that every chunk of code you write will result in several contributions to a growing library of appliance code. I merely suggest that the next version is easier to alter, and less wedded to a particular microprocessor. That's often payoff enough.

The simple fact is that moving from

assembly language to a higher-level language is the biggest single step you can make toward improving code quality and overall productivity. It doesn't take the place of learning better design techniques, but it sure assists in the process. It doesn't ensure that you will test any better, but it usually makes the testing process go faster. And it doesn't guarantee better code hygiene, but it helps enforce good practices if you choose to follow them. Switching languages is the nearest thing to a mechanical step you can take toward improving the overall program-development process. The fact that it also has such a big payoff makes it an obvious thing to do—at least if you buy what I've said so far.

Pros and cons

So why haven't all small-to-medium-sized embedded projects switched to

C? After nearly 30 years, we now have a multitude of compilers and supporting tools to choose from. Just look through the ads in this magazine, if you haven't done so already. C compilers and cross compilers now come in all shapes and sizes, for practically every processor in widespread use. It is not the want of tools that should be holding people back.

The fact is that you, the typical reader of this magazine, don't need to be convinced. I've seen the reader polls, year after year. I know that I'm preaching to the converted. The typical *ESP* reader is overwhelmingly likely to be programming in C, C++, or maybe Java these days. Only a small and dwindling fraction lists assembly language as a principal working environment.

In part, I'm arguing the case for you holdouts. But to a larger extent my remarks are aimed at the rest of you who still have colleagues working less efficiently with older tools. Any shop still using tools upwards of 40 years old has obviously been exposed to the arguments for change. It will take some convincing to beat down whatever resistance has hardened over the years. But friends don't let friends write assembly language—not if a sensible choice is available off the shelf.

The most common source of resistance is a simple fear of change. Many programmers work in shops that remain in semipermanent crisis mode. They move from one rush job to the next. In between, any "spare" time goes toward patching up code in the field, to keep it going until the next release gets out the door. If you're overworked, under budgeted, and working to silly deadlines, any change in practice appears insurmountable.

An outsider can observe, accurately enough, that the use of old tools is a major contributor to the perennial shortage of resources, but that seldom helps. All that stimulates is a nonsense defense along the lines of, "We don't have time to stop everything and rewrite all our code in C." Or maybe,



**Having
problems
landing
qualified
embedded
designers?**

**Advertise in Embedded
Systems Programming's
career section!**

**Contact Sam Louis
415/278.5223**

"That may be okay for you, but we have severe size and performance requirements that we can only be sure to satisfy by sticking with assembly language." Both arguments are straw men, of course, but they take some effort to wear down.

It is also worth noting that there are still good reasons for *not* converting from assembly language to C, or some other high-level language. The reason I accept most readily is size. Not all embedded systems have to be as smart as a VCR or a rice cooker. We're not quite down to the level of replacing every cam and spring with a microprocessor, sensor, and transducer, but we're getting closer all the time. Articles in *ESP* may dwell on the problems of programming 16- and 32-bit processors in C++, but plenty of systems are still out there that really can't support a Standard C compiler. The processor may be too simple, with insufficient support for 16-bit arithmetic, or with inadequate support for a call stack. Or memory may simply be too small, measured in hundreds or thousands of bytes, rather than tens of kilobytes or larger.

For these systems, I would still investigate whether some dialect of C is available. Quite a few subset compilers are available for eight-bit processors. Barring that, I would look for a "C-like" compiler that offers practically any degree of data typing and control-flow structuring. But beyond a certain point, I would just give up and stick with assembly language.

Another reasonably good argument for not using C is when performance really does push the limitations of the hardware. But in these cases, I always cite two caveats. First, no system is likely to be robust if it uses more than 70% of any resource, be it processor time or heap storage. If you're that close to the line, you're probably better off moving to the next more powerful processor or the next larger RAM or ROM. Development costs also soar as you approach saturation. Given the

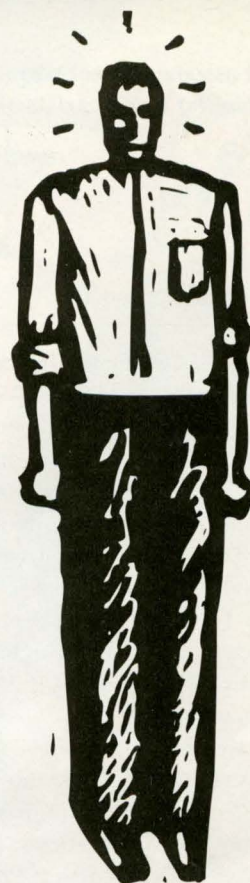
steadily falling costs of hardware, you have to be careful to avoid false economies in this area.

The second caveat is that a program always has just one or two true hotspots, the places where performance makes all the difference. For every line of performance-critical code, you typically write yards of vanilla code that just has to be correct, not fast. It is faster, easier, and safer to grind out that pedestrian code in a high-level language than in assembly language. In fact, I have yet to see a system that can't be brought to the ultimate in performance by rewriting just a few small bits in assembly language, leaving the vast bulk in C.

The same is true for code that uses funny instructions, such as for direct input/output, or for performing magic DSP operations. Remarkably few places need to step outside the normal bounds of C code. If the compiler doesn't permit the necessary escapes, you can usually package the funny lumps as C-callable functions. But the need for a dozen special instructions is no excuse for writing thousands of lines of assembly language that can be better expressed in C.

Still other valid reasons exist for not moving away from assembly language. If you have a largely stable product, it's hard to justify rocking the boat. If you have to live with a certification process, such as for air worthiness or product safety, you may face a huge cost to recertify even the most obvious and limited of rewrites. And if you are far enough into a project, you dare not risk schedule slippage, or even failure, by introducing new technology along the way. Transitions *always* cost some amount of time and money. Whether you can recoup either or both in time depends strongly on where you are in a project life cycle. Only a high-level language fanatic would ignore real-world factors such as these and insist on converting over immediately. I try not to be such a one.

DON'T FRET!



Reprints are easy to get!

Full of interesting articles and valuable industry news, **Embedded Systems Programming** has information you'd like to share. Reprints are great for trade shows, marketing kits, and presentations.

Contact Sherry Bloom for details.

tel: 415/905-2701

fax: 415/975-3111

Email:

sbloom@mfi.com

Don't assume a direct correlation between price and quality. Free software of the highest quality exists alongside expensive dogs. Read reviews and talk to your friends. But don't be too leery.

A practical approach

So how do you make the change, if you become so inclined? Clearly, you do not stop the world long enough to rewrite everything in C. Nor do you rewrite a critical program completely in C and hope that an optimizer will make it run fast enough. The only sensible way to introduce new technology of any kind is in stages. I encourage people to begin with one or more non-critical projects, phasing in the use of C a bit at a time.

I have described an ideal high-performance program as one written almost entirely in C, with maybe a few assembly language functions thrown in. But you can also do it the other way 'round. Say you need to add a module to an existing system. Ask yourself, can it be written in C, at least in principle? If the answer is yes, then suspend for the moment, as best you can, any fears that it won't be fast enough or small enough.

Write the code as one or more C functions, to be called from the existing code. Get it working and measure the performance and code size. My bet is that both will be acceptable right off the mark. But if not, first consider rewriting the C code before you give up. Instrument it and see where the problems lie. Often a small change in data structure, or algorithm, will make all the difference.

The most important thing is to be willing to rewrite code. When I first start working with a new programming language, I rewrite code repeatedly, often a dozen times or more. Even

when I become a nominal expert in a language, I usually rewrite code three or more times before I'm satisfied. If you think that's a luxury you can't afford, I beg you to reconsider. I make a living selling code commercially, just like many of you readers out there. After over three dozen years of doing so, I am convinced that I can't afford to develop software any other way.

The point is that programming is an art that you can only learn by doing. The only program you know how to write is one you've already written at least once. In the early days of learning a language, you see better ways of expressing code on a daily basis. Until you get the idioms down, you're not likely to produce code that's maintainable over the long haul.

If you form the habit of writing new pieces in C, you will gradually displace the old stuff. As luck would have it, the bits you're most likely to rewrite are the ones most in need of rewriting. Five years from now, you may notice that you still have several chunks of eminently stable assembly language code in the product. If you're so fortunate, don't fret about leaving it there for still more years to come. Just be grateful that the more volatile components are steadily becoming easier to maintain.

As a final note, I tell would-be converts that it's much easier to *be* a C shop than to become one. One of the hardest parts of converting over is picking the right compiler. (The next hardest is getting it to compile a program that prints "hello world," but

that's another story.) You can get some pretty inexpensive compilers these days, even cross compilers for developing embedded systems. If you decide to get one, however, make sure you can afford it. You'll have to work harder to get support, and you may have to fix bugs in the compiler and tool chain yourself. Cheap software can be a bad bargain.

I hasten to add that expensive software can also be expensive. I won't name names, but I have, over the years, worked with commercial cross compilers that can barely justify their license fees. Don't assume a direct correlation between price and quality. Free software of the highest quality exists alongside expensive dogs. Read reviews and talk to your friends. But don't be too leery. The really bad stuff is eventually exposed and fades from the scene, thanks to all the competition out there.

As a final final note, I emphasize once more that you can still make a case for writing some code in assembly language. But the number of such applications is dwindling. C has been justly accused of being no more than portable assembly language, but that's not such a harsh criticism if you happen to like assembly language and know its virtues. I believe that mixing assembly language and C is the approach of choice for many small to medium embedded systems. For larger systems, you should consider even more structured languages such as C++ and Java. But that's still another story.

esp

P.J. Plauger is the author of the standard C++ library shipped with Microsoft Visual C++. His latest books are The Draft Standard C++ Library and Programming on Purpose (three volumes), both published by Prentice Hall in Englewood Cliffs, NJ.

EMBEDDED SYSTEMS PROGRAMMING (ISSN 1040-3272) is published monthly, with an additional issue published in September, by Miller Freeman Inc., 525 Market St., Ste. 500, San Francisco, CA 94105, (415) 905-2200. Please direct advertising and editorial inquiries to this address. SUBSCRIPTION RATE for the United States is \$55 for 13 issues. Canadian/Mexican orders must be accompanied by payment in U.S. funds with additional postage of \$6 per year. All other foreign subscriptions must be prepaid in U.S. funds with additional postage of \$15 per year for surface mail and \$40 per year for airmail. POSTMASTER: All subscription orders, inquiries, and address changes should be sent to EMBEDDED SYSTEMS PROGRAMMING, P.O. Box 1278, Skokie, IL 60076-8278. For customer service, telephone toll-free (888) 847-6177. Please allow four to six weeks for change of address to take effect. Periodicals postage is paid at San Francisco, CA and additional mailing offices. EMBEDDED SYSTEMS PROGRAMMING is a registered trademark owned by the parent company, Miller Freeman Inc. All material published in EMBEDDED SYSTEMS PROGRAMMING is copyright © 2000 by Miller Freeman Inc. All rights reserved. Reproduction of material appearing in EMBEDDED SYSTEMS PROGRAMMING is forbidden without permission. EMBEDDED SYSTEMS PROGRAMMING is available on microfilm/fiche from University Microfilms International, 300 N. Zeeb Rd., Ann Arbor, MI 48106, (313) 761-4700.

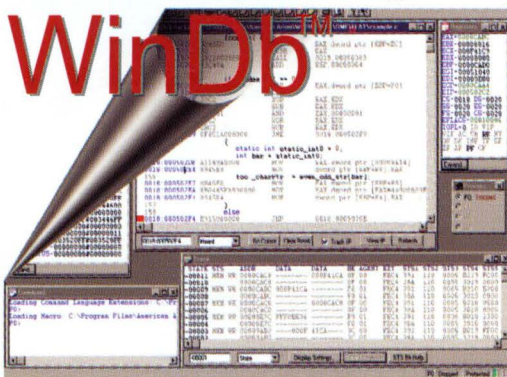


It's hard to compete without the right tools.

You can't afford to get left at the gate if your new Intel processor based system is going to finish in the money. But helping you finish in the money is what we're all about.

Ever since 1991, American Arium has been working closely with Intel to create powerful new development tools for each new Intel processor.

So whether you've designed an embedded system around a Pentium® processor, or a server using a Merced processor, we can provide you with the most



advanced in-circuit emulators in the business. Also...

- they're available with early silicon
- they're affordable
- we provide in-depth tech support and
- there's an upgrade path

Don't risk finishing back in the pack. Let us help get you to market ahead of schedule and under budget.



14811 Myford Road, Tustin, CA 92780
Ph: 714-731-1661 Fax: 714-731-6344
e-mail: info@arium.com

For more information on any of our in-circuit emulators, visit...

www.arium.com

8051

68HC11

Z80

Z180

68HC12

Mcore

683xx

PowerPC

68K

x86

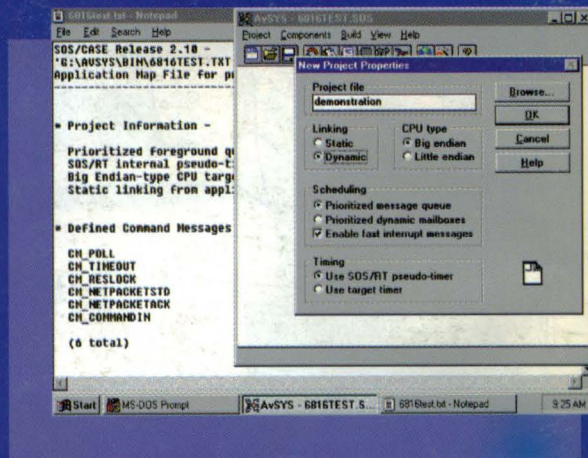
68HC05

ColdFire

And
Many
More

You Need AvSYS.

With AvSYS Real Time Operating System, your project starts two weeks ahead of schedule. Don't reinvent the wheel when you can have the power of pre-written and pre-tested code at your fingertips. AvSYS makes your code organized AND portable, creating a cohesive standard within, and across, projects. Handle interrupts, task-switching, and timing with ease. Save yourself weeks of development, testing, and maintenance. AvSYS is the smart choice for embedded development.



**Free Demo
Available!**

AvSYS: The Complete Solution

- ❖ Getting started is a snap with our code-generating IDE
- ❖ 100% ANSI C Source Code Included
- ❖ Fully Scalable from micro-kernel to full RTOS
- ❖ Tiny Footprint
- ❖ Ultra-fast Context Switching
- ❖ Flexible Network Manager
- ❖ Completely Portable
- ❖ Royalty-free

**Best of all, Avocet makes
your life easier by meeting
ALL your tool needs**

Avocet: The Complete Solution

- ❖ Over 20 Years of Embedded Tools Experience
- ❖ Compilers, Assemblers, Simulators, RTOS, In-Circuit Emulators for Hundreds of Processors
- ❖ Expert Technical Support Staff
- ❖ We will get you any tool you need

www.avocetsystems.com

sales@avocetsystems.com

(800) 448-8500

AVOCET SYSTEMS, INC.