DVME-601

User Manual

for DATEL's
Intelligent
VME bus A/D Board

DATEL, Inc. 11 Cabot Boulevard, Mansfield, MA 02048-9984
(508) 339-3000

January, 1993 UM-DVME-601

CAUTION

Do not apply power to this board until you have read this manual

Unpacking

This product should include a DVME-601 board in a static-protective bag, and a User Manual. Upon receipt, carefully remove the board from the anti-static bag and visually inspect the board for physical damage which may have occurred in shipment. In the event of damage, save all packing materials for inspection by your carrier to validate shipping damage claims. Should the board need repair at a later date, it may be safely returned in its original packing materials. If you do not intend to use the board immediately, store it in the anti-static bag until use.

CAUTION - Always store the board in the anti-static bag

This board contains components that can be damaged by static discharge and should be handled with caution. Make sure your body is earth grounded before handling the board, especially in dry, low humidity conditions. Avoid board contact with clothing having a high synthetic content. Avoid contact after walking across carpeting. Before you first touch the board, continuously hold the VME chassis it will be installed into. In severe static discharge environments, consider grounding wrist straps or other protective measures.

Warranty

This board was precision manufactured with the highest quality methods. DATEL warrants that this product was manufactured free of defects in workmanship or materials under normal use and service as described in the literature for this product. Obligations under this warranty are limited to replacing or repairing, at DATEL's option, any of said products, at DATEL's factory or facility, transportation charges prepaid, and which are, after examination disclosed to the satisfaction of DATEL to be thus defective, for a period within one year of shipment. This warranty shall not apply to any such products which have been repaired or altered except by DATEL or which have been subject to misuse, negligence or accident. In no case shall DATEL's obligation exceed the original purchase price. The aforementioned provisions do not extend the orignal warranty period of any product which has either been repaired or replaced by DATEL. This warranty does not contain a guarantee, either expressed or implied, of merchantability or fitness for a particular purpose. DATEL cannot assume liability for damage or loss as a result of the use of this product. It is the customer's sole responsibility to determine if this product is suitable for his or her application.

Disclaimer

All information contained in this document has been carefully examined and is believed to be entirely accurate. However, DATEL assumes no responsibilities for errors or omissions. DATEL reserves the right to make changes to this manual without prior notification in accordance with DATEL's policy of product support and improvement.

Proprietary Notice

The information and the design of this product are the exclusive property of DATEL. This entire document is proprietary to DATEL and shall not be disclosed to third parties without the express written consent of DATEL. This information is not to be photocopied, transcribed, recorded, communicated or reproduced by any means without permission. The foregoing does not apply to vendor proprietary rights or rights under the patents of third parties. Use of this product does not convey a license, either expressed or implied.

Quick Reference Guide

Serial Port Monitor Commands

Carriage return terminates all commands and parameters. See section 3. Exit Modify commands with ".<cr>"

Command	Function
H	Display Help menu.
RD	Display all 68010 internal registers.
RM <reg>	Display/Modify cycled CPU registers.
RI [offset](reg)	Display/Modify offset indirect memory.
MD <start adrs> [<end adrs>] or [,<count>]	Display block of memory in hex and ASCII.
MM <start adrs>	Display/Modify memory. ".<cr>" to exit.
BF <start adrs> <end adrs> <data>	Fill memory with "data".
BR <break adrs> [,<count>]	Set break point and optional count.
BR	Display break point and count.
NOBR	Delete all break points.
T [<start adrs>] [,<count>]	Trace "count" instructions (default = 1) beginning at current PC or "start adrs".
G [<start adrs>]	Execute from PC or adrs if EXEC ON.
EXEC ON	Go to Exec level (Awake) for DPR commands.
EXEC OFF	Go to MONITOR level and ignore DPR.
Control-C	If at EXEC level, go to MONITOR level but stay Awake. Halts running program.
AD <start addr> [,<final adrs>]	Display continuous A/D conversions. ^S to pause, ^Q to resume, <cr> to stop.
SRDL	Start serial port download. ^X to abort.
Control-X	Hard reset to Monitor. Abort locked CPU.
TC	Load. chan. adrs. reg. with 00->FF bytes
CT <adrs>	Write continuous 00->FF byte at memory.

Table of Contents

Section 1 - General Information

Section 2 - Board Configuration and Installation

Section 3 - I/O Connections and the Monitor

Section 4 - Programming Introduction

Section 5 - Programming the Executive

Section 6 - Local Programming

Section 7 - External Triggering

Section 8 - Theory of Operation

Section 9 - A/D Section Calibration

Important: For channel expansion inputs, connect the solder gaps. See Section 3.2

GENERAL INFORMATION

1.0 Introduction

1.0.1 Scope of This Manual

Thank you for your interest in DATEL's DVME-601 A/D coprocessor board. This manual contains complete information to configure, install, program and use the DVME-601. Diagnostic, theory of operation and calibration information is included as well as a full set of schematic and assembly drawings. For customers, DATEL also offers the source code for the Monitor/Executive EPROM firmware on separate media.

Preliminary trouble-shooting may be accomplished with this manual but extensive component level repairs should be referred to DATEL. Users with critical downtime applications should consider stocking whole spare boards.

1.0.2 Who This Manual Is Intended For

DVME-601 users should be familiar with all aspects of developing a high performance data acquisition application. This includes selecting, installing and connecting sensors to the DVME-601, complete operation of the host VME computer, and software programming. The user should clearly understand the arithmetic procedures for handling A/D data, possible error sources (both hardware and software) and system timing required. Some knowledge of digital logic, microprocessors and precision analog circuits will prove helpful. Although the DVME-601 contains extensive programs in its EPROM memory, the user must still write computer programs in the host to control the DVME-601 and transfer A/D data blocks to system memory. If any local DVME-601 programs are to be developed, the user must be proficient in 68000/68010 assembly language even if a high level language is used. The user must also be skilled at using the program development tools and operating system of the host computer. Software examples given in this manual assume that the user understands 68010 programming.

1.0.3 Related Information

Besides this manual, the following information should be consulted:

DATEL DVME-601 Data Sheet

DATEL DVME-601 EPROM Source Code and Related Programs (available on floppy disks and other media for various computers - contact the home office. These disks contain valuable program examples).

MOTOROLA M68000 Microprocessor Programmer's Reference Manual (Contact Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 07632 for the latest edition. This includes 68010 information.)

Users should also have the full set of hardware, program development and operating system documentation available for their host VME computer.

1.0.4 Equipment Requirements

Besides the DVME-601 board, you will need the following:

An analog signal source such as DATEL's DVC-8500A Voltage Calibrator or a group of input sensors.

[Optional]: A second asynchronous ASCII terminal/keyboard (RS-232-C, 9600 baud, 8 data, no parity, 1 stop) is very desirable for calibration and initial testing, even if you do not intend to write local DVME-601 programs. It will connect to the DVME-601 front panel. This terminal is in addition to your host terminal. An RS-232-C crossover cable will be needed. This terminal is mandatory for debugging local DVME-601 programs.

[Optional]: A VMEbus extender board.

[Optional]: Various probes and test equipment such as a pulse generator to simulate external A/D start triggers.

[Optional]: If you write local programs and load them into the DVME-601's EPROM's, you will need an EPROM programmer and cabling which can accept 27C256 or 27C512 EPROM's. You may also want an ultraviolet erasing lamp. A communication or copy program is required in your host to transfer your DVME-601 S2 records to the EPROM programmer.

1.0.5 Implementation Procedure

The general outline for implementing an application using the DVME-601 is shown below. This procedure will be expanded throughout the rest of this manual.

1. Study your host computer and its documentation to decide how the DVME-601 will be integrated (board slot, memory addressing, interrupts, Operating System interface, etc.).

2. Configure the on-board hardware (jumpers, PGA gain, etc.) if it should be different from the factory settings.

3. Install the board and connect the optional terminal. Verify that your host operates normally after the installation. Do preliminary testing and command familiarization. Connect analog input signals.

4. Develop and debug your software for both the host and optional local programs. Reprogram the EPROM if desired.

5. Document your application so that it can be maintained and enhanced and for application questions.

1.1 Description

Designed for today's VME environment of real-time operating systems and busy host processors, DATEL's DVME-601 is a high performance A/D data acquisition system integrated with a local 68010-based single-board microcomputer. The DVME-601 smart A/D board performs simultaneous analog data collection while concurrent VME host processing continues. Unlike dumb A/D boards, the DVME-601 will automatically collect scans without delaying other host software tasks. When A/D data is ready, the 601 can interrupt the VME host so that the host can memory transfer previous scans without stopping collection of the next scan.

Typical applications for the DVME-601 include high speed process control loops, analytical instruments, research laboratories, vehicular data recorders, ATE equipment and communications testers. The block-oriented, interrupt-controlled, memory transfers of A/D scans to the host are particularly suited to Digital Signal Processing applications in acoustics, sonar, high speed mapping, seismology, medical imaging, graphics, array processing, FFT's and waveform analysis. Using the single-channel fast-throughput mode and 2 microsecond converter, true 12 bit speeds over 250,000 samples per second to memory are available (single-channel, gain=1).

The data acquisition section includes an analog input multiplexer with 16S/8D local channels expandable up to 256 total channels using DATEL's slave MUX boards. An Instrumentation Amplifier is included which may be resistor-programmed by the user up to a gain of 1000. A choice of pluggable A/D Converter modules is offered on five different models. Resolution from 12 to 16 bits is available with 12-bit conversion speeds down to 2 microseconds.

A/D conversion may be started by external trigger, internal programmable timer, by local programs and indirectly by host command. All of these start modes are managed by on-board EPROM firmware. A DC/DC converter supplies low-noise regulated power to the data acquisition section.

The primary interface between the 601 and its VME host is a 64 Kb dual ported Random Access Memory (DPR). The DPR is used for commands, subroutines and parameters, control/status bits, A/D data blocks, optional downloads of user programs, and bidirectional interrupts between the 601 and the host. The maskable interrupt to the VMEbus normally occurs after A/D scanning but may also be commanded from a local user program.

Executive firmware included in the EPROM offers many ways to manage the DPR including swapped buffer ("ping-ponged") A/D scan transfers while the host reads the alternate buffer. The user may run the 601 either in the "no-programming" mode or may load and execute his own programs. The no-program mode uses fast A/D routines supplied in the EPROM. The firmware also includes a serial port Monitor to develop optional user programs.

The power and flexibility of the 601 can be enhanced with user-written software. As a high performance, general purpose microcomputer, the DVME-601 is ideal for automatic A/D data collection and arithmetic preprocessing of A/D data before sending it to the host. By transferring preprocessed results rather than raw data, total system performance can be increased while the host continues with disk, display or control activities. Programs may be developed in the host, saved on disk, then downloaded to local RAM via the DPR or serial port. The pluggable EPROM may be reprogrammed by the user or by DATEL under special order. Any language may be used such as Assembly, BASIC, FORTRAN or C if it can be compiled to 68010 code.

The local microcomputer consists of an 8 MHz MC68010 microprocessor, 64 Kb of pluggable EPROM (socketed to 128 Kb), 64 Kb of DPR and 64 Kb of private RAM. Total local storage of A/D data may approach 60,000 samples using the DPR plus private RAM if no other program is using RAM space.

A Multifunction I/O Peripheral using a programmable 68901 controller is also included offering an interrupt controller, an RS-232-C serial port, 4 timer/counters, and 5 I/O bits. Some of the timers and the serial port are used by the Monitor/Exec firmware but may be reprogrammed by the user including external interrupts. A second RS-232-C serial port is available using a software UART and the I/O bits. A front panel green LED lamp is illuminated to confirm power-up selftest and may be programmed by the user for alarms, etc.

Using the serial port, a 16 MHz clock source and a +5V dc power supply, it is even possible to operate the DVME-601 in stand-alone mode, not connected to the VMEbus. Commands and A/D data pass through the port up to 19.2 kilobaud with user-written programs.

The board uses +5V dc at 3 Amps and ±12V dc at 5 mA (typical) from the VMEbus. Connections are made only to P1 (P2 is not used to assure compatibility with most hosts). Data transfer is 16 bits wide. The DVME-601 is a D16 A24 slave board using 24 address lines and 6 address modifier lines. The board occupies 64 Kb of host memory. A single interrupt to the host asserts a programmable interrupt vector. Three front panel D connectors are provided for local analog inputs, analog slave-MUX channel expansion and for the Serial/Parallel/Timer I/O.

The DVME-601 is supplied on a 9.19"W x 6.3"D x 0.6"H (233,5 x 160 x 15,2 mm) 6U board. It includes a comprehensive User's Manual with programming information. Access to the EPROM source code in several formats is available to customers on special request.

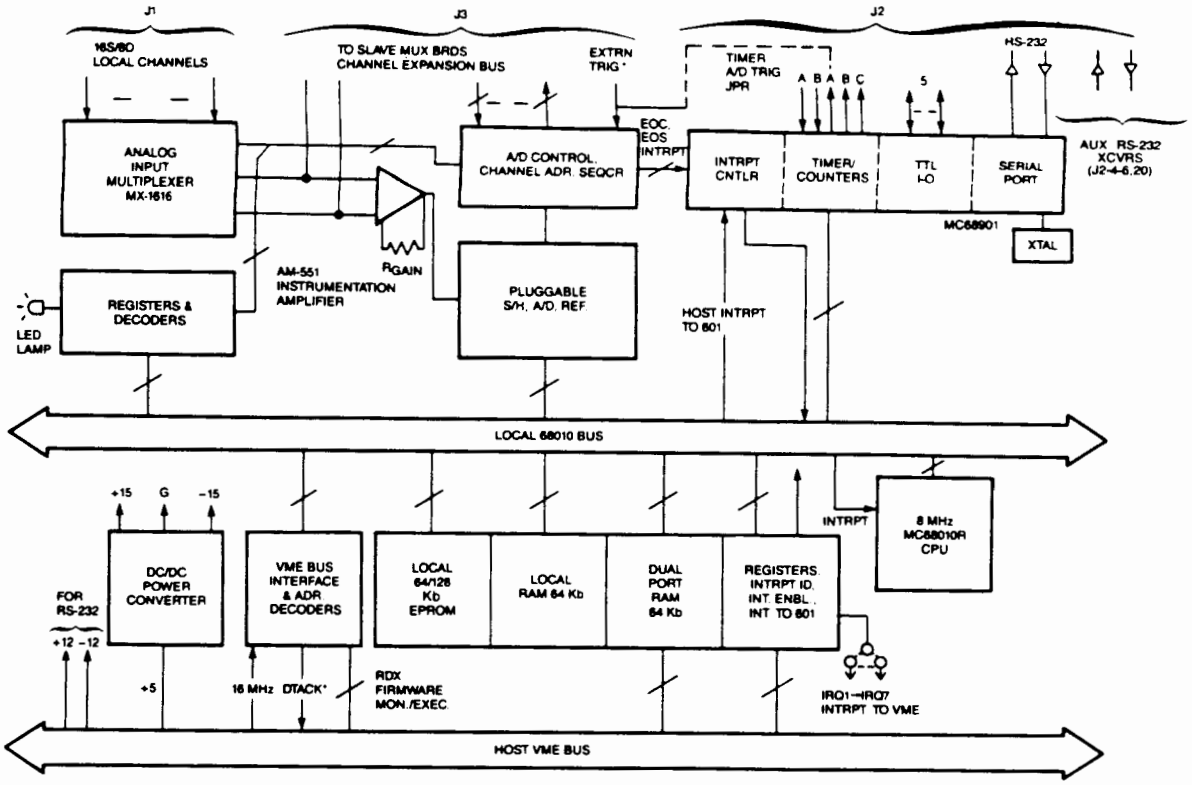


Figure 1.1 DVME-601 Block Diagram

1.2 DVME-601 Specifications

Specifications are typical @ +25 °C, gain = 1, unless noted.

Data Acquisition Section

Number of On-board Channels	16 single-ended or 8 differential inputs, non-isolated, jumper selectable.
Analog Channel Expansion	Up to 256 total channels, local and remote using DATEL's channel expansion bus and slave Multiplexer Boards. Channels may be mixed single-ended or differential on slave MUX boards. (10 slave MUX boards maximum)
Input Voltage Range	±10 Volts full scale. (±5V, 0 to +10V, and 0 to +5V may be jumpered or special-ordered. See Figure 2.6).
Common Mode Voltage Range	±10V, maximum, non-isolated.
Common Mode Rejection	80 dB, dc to 60 Hz, CMV = ±10V, with 1 Kilohm source unbalance, gain = 1.
Input Bias Current	±200 pA

Overvoltage Protection	±30V dc, maximum sustained
Input Impedance	Power on: 1000 Megohms, differential or to ground. Power off: 1.5 Kilohms
A/D Output Coding	Left-justified bipolar 2's complement, jumperable to bipolar offset binary or unipolar straight binary for DVME-601A. Other models are bipolar 2's complement or offset binary.
Instrumentation Amplifier Type and Gain Range	AM-551 supplied as gain = 1. May be resistor-programmed by user up to gain = 1000 with increased settling delay.
Instrumentation Amplifier Settling Delay (gain=1)	3 microseconds to 0.01% of FSR (The Inst. Ampl. may be by passed for faster single-ended throughput.)
Adjustments	Inst. Ampl.offset, A/D offset and A/D gain.

A/D-S/H Resolution and Conversion Period Options: (Uses pluggable modules)

Model	Resolution	Convert time	Thruput to RAM*
DVME-601A	12 Bits	20 microsec.	40 KHz
DVME-601B	12 Bits	4 microsec.	114 KHz
DVME-601C	16 Bits	35 microsec.	25 KHz
DVME-601D	16 Bits	400 millisec.	2.5 Hz
DVME-601E	12 Bits	2 microsec.	see notes
DVME-601F	14 Bits	4 microsec.	100 KHz

*The typical throughput rate is an aggregate time within a multichannel scan and does not include subroutine setup time. The rate includes times for channel sequencing, MUX, Inst. Ampl., S/H acquisition/ settling, A/D conversion, and 68010 software times. The polled-EOC STSNSC subroutine is used, triggering A/D conversion on each data read. Higher single-channel speed is available using the "fast-throughput" mode (delayed DTACK✱). The DVME-601E offers 300 KHz in a long burst to local RAM, single channel, gain=1. In multichannel, all modes require at least 6 microseconds (gain = 1) from channel sequencing to A/D start plus the A/D conversion time. Data transfer time may overlap A/D conversion.

System Performance

Specification	A/D type				
	601A 12 bit, 20 µS	601B,E 12 bit, 2 or 4 µS	601C 16 bit, 35 µS	601D 16 bit, 400 mS	601F 14 bit, 4 µS
Accuracy, min.	0.025% of FSR	0.05% of FSR	0.01% of FSR	0.0063% of FSR	0.01% of FSR
Nonlinearity, max.	1/2 LSB	1/2 LSB	2 LSB	2 LSB	2 LSB
Zero tempco, max.	±20 ppm/ °C	±20 ppm/ °C	±20 ppm/ °C	±10 ppm/ °C	±15 ppm/ °C
Gain tempco, max.	±20 ppm/ °C	±20 ppm/ °C	±20 ppm/ °C	±10 ppm/ °C	±15 ppm/ °C

External A/D Start Trigger	Negative-going TTL input with 4.7 Kilohm pullup to +5V. Pulse width 100 nS min., 2 microsec. max.
Local Pacer Clock	Software programmable to cause A/D Start Timer either an A/D scan start or a single conversion.
Pacer Clock Interval Range	3.255 microseconds to 41.667 mS (supported in firmware) using one timer. Up to 3.03 hrs. using 3 cascaded timers.
	Local Microcomputer
CPU Type and Clock Speed	MC68010R8, 8 MHz (CPU clock is derived from 16 MHz bus SYSCLOCK).
Local data bus width	16 Bits
Local Read/Write Memory	64 Kilobytes static RAM (no VMEbus access)
Local Read-only Memory	UV-erasable EPROM, 64 Kilobytes supplied as two 27C256's but is socketed for two 27C512's totalling 128 Kb. The EPROM contains Monitor/Executive firmware.
Dual-ported Read/Write Memory (VMEbus and local access)	64 Kilobytes
Front Panel LED Lamp	Green LED lamp is lit if power-up local CPU selftest succeeds. Thereafter, the lamp may be programmed by user software for alarms, etc.
	Peripheral I/O Controller
Controller Type	MC68901 multifunction peripheral, crystal-controlled, 2.4576 MHz, user-programmed.
	Interrupts
Local Hardware Interrupts	A/D End of Conversion, to 68010 CPU (Maskable) A/D End of Scan, VMEbus host command request
Local Software Interrupts	Any timer count reached or (Programmable) I/O bits 0-4.
	Digital I/O
Number of I/O Lines	5 lines, individually programmable as inputs or outputs or interrupts.
Logic Levels	TTL levels, 1 load max. with 10 Kilohm pullups to +5V.
	Timer/Counters
Number of Timers	4 8-bit timers with prescale up to divide-by-200.

Timer Outputs	3 outputs, (Timers A,B,C), 1 TTL load max. The Timer D is the USART baud clock but may be reprogrammed.
Timer/Counter Inputs	2 inputs, TTL levels with 10 Kilohm pullups to +5V.

Serial Port

Number of Serial Ports	1 USART, full duplex, RS-232-C levels, DTE pinout. Aux. RS-232-C I/O may be configured.
RS-232-C Handshakes	DTR, DSR, RTS, CTS program- by the user. (See J2 pinout).
Modes	Synchronous or asynchronous.
Number of Stop Bits	0, 1, 1.5, 2
Number of Data Bits	5, 6, 7, 8
Parity	Odd, even, none for receiver. Transmitter is user-coded.
Baud Rates	Up to 19.2 Kilobaud.

[EPROM Monitor firmware uses the serial port and timer D at 9600 baud, 8 data, no parity, 1 stop. A 4800-baud software UART is formed with I/O bit 0 for optional serial S record downloads. Firmware also uses Timer A as an A/D start clock. All functions may be reprogrammed by the user.]

VMEbus Interface

Standards Compliance	IEEE P1014/D1.0
Data Bus Width	16 Bits
Address Bus	A24 D16 slave, 24 address lines, A23 - A01, plus 6 address modifiers, jumper selected.
Address Modifier Codes	39h, 3Ah, 3Dh, 3Eh jumperable. (see jumper information)
Architecture	Dual-ported 64 Kb block mapped on 64 Kb boundaries. One address (force interrupt to 601) is write-only. All others are Read/Write. (See memory map).
VME Bus Interrupter	1 line, jumper-selectable IRQ1* thru IRQ7*. Asserts one interrupt ID code which is programmable from the host. Interrupt is generated by a 601 local register write and is maskable by a host-writable register.
Data Transfer	16 bits using P1. Generates DTACK* derived from 16 MHz bus clock.

Connectors

VMEbus, P1	96-pin male DIN connector. No P2 connector is used.

Local Analog Input, J1	25-pin DB-25S female on front panel.
Multifunction I/O Peripheral (68901), J2	25-pin DB-25S female on front panel.
Analog Input Channel Expansion Bus, J3	25-pin DB-25S female on front panel, compatible to DATEL DVME-64X series MUX boards.
	Miscellaneous
Power Required	+5V dc ±5% @ 3.0 Amps max. (DVME-601A) or 3.1 Amps max. (DVME-601B,C,D,E,F) and ±12V dc @ 10 mA max. from VMEbus. The 12V power is only required if the serial port is used. A local ±15V dc DC/DC converter is included for linear circuits.
Operating Temperature	0 to +60 °C
Storage Temperature	-20 to +80 °C
Relative Humidity	10% to 90%, non-condensing.
Outline Dimensions	Double-height VME, 6U outline. 9.19" W x 6.3" D x 0.6" H, (233,5 x 160 x 15,24 mm).
Weight	17 ounces (482 grams)

Allow 20 minutes warmup time for model DVME-601F.

BOARD CONFIGURATION AND INSTALLATION

2.0 Determining Your Host Environment

Before installing the board in your host computer, you should consult your host documentation. Installation must not disrupt the normal operation of your host and must be compatible with its resources. If your development environment differs from the target application, you should be aware of the differences.

DVME-601 host compatibility involves two major areas:

2.0.1 Host Memory Map Compatibility

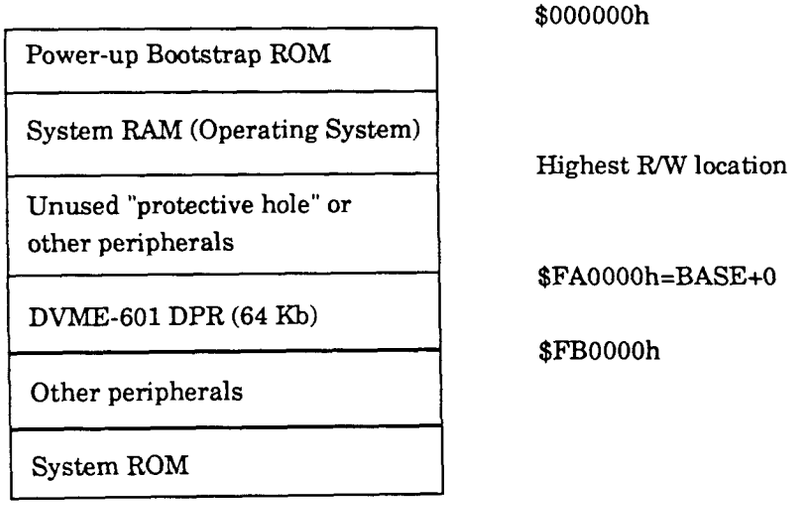
The 64 Kilobyte memory block occupied by the board must not conflict with other system memory. This has two consequences:

1. The board's 64 Kb dual port RAM cannot overlay or respond to the same address codes as Random Access Memory (RAM) or other memory-mapped peripherals in the host.

2. On host power up, many operating systems test for how much contiguous read/write memory there is, starting from low memory. If the DVME-601 is mapped adjacent to system RAM used by the operating system, memory location BASE + $0FFFC will be detected as the top of contiguous RAM. This is because this location is actually a DVME-601 write-only memory-mapped hardware register and will be the first location which fails read/write memory testing by the host. Some operating systems attempt to place the machine stack just below this area, which would be located at the top of DPR. This conflicts with other DVME-601 hardware registers and the DPR command area.

Other operating systems may attempt to read memory size switches or PROM's set by the computer manufacturer. These should not be changed to accomodate the DVME-601.

A recommended procedure is to locate the DVME-601's BASE address away from system RAM, separated by an area which will fail operating system memory testing. (This area should not have anything which responds to a memory read/write access). This "protective hole" will make your system memory map appear similar to figure 2.0. (Factory supplied addressing is shown). Positioning the DVME-601's BASE address relative to the host is done by setting the base address switches, discussed in the next section.



Default factory-jumpered BASE addressing is shown for the DVME-601.

Figure 2.0 Typical Simplified Host Memory Map

2.0.2 Host Interrupt Compatibility

The other compatibility area involves VMEbus interrupts. If you plan to make long multiple A/D scans which are asynchronous with other complex host activities, interrupts are a desirable way to periodically resynchronize the entire system and maintain system bandwidth. Interrupts require the following decisions:

A. The Interrupt Request line to VMEbus (IRQ1 through IRQ7) must be jumpered. This determines interrupt priority. This is discussed in following sections.

B. A vector address must be loaded by your host into the DVME-601's ID register (DPR location BASE + $0FFFA) after host power-up. This address is used by the CPU (after 2 left shifts) to develop the interrupt service routine (ISR) address. The ISR is either an absolute address or an offset from the CPU's Vector Base Register.

C. Programs are required in your host to respond to DVME-601 interrupts, process commands and transfer A/D data.

Your target application is likely to handle interrupts in one of two ways:

A. If you are NOT using a multitasking Real Time Operating System (RTOS), the interrupt controller in your host must be loaded with a vector address pointing to code to handle interrupts from the DVME-601. Obviously you must also have developed such code and it must be resident in memory.

B. If you ARE using an RTOS, interrupts are handled symbolically and you probably will not have to do low level interrupt controller programming. You must still write code to process DVME-601 commands and transfer data plus arbitrate with the RTOS. You will need to decide a priority for the DVME-601, how it communicates with other system tasks and how its status is managed. RTOS programming is beyond the scope of this manual.

Be aware that processing interrupts through an RTOS can take extra time because of overhead. With multiple random system interrupts, this delay is not always predictable or easy to measure. Unless interrupts are temporarily masked off during a long A/D memory fill, RTOS usage may not be suitable for very high bandwidth applications.

Most users will develop their DVME-601 application on a DOS-based development system. Target applications which execute out of PROM without a DOS may have many differences in memory mapping, interrupts, etc. Be aware of these differences in transferring from development to target environments.

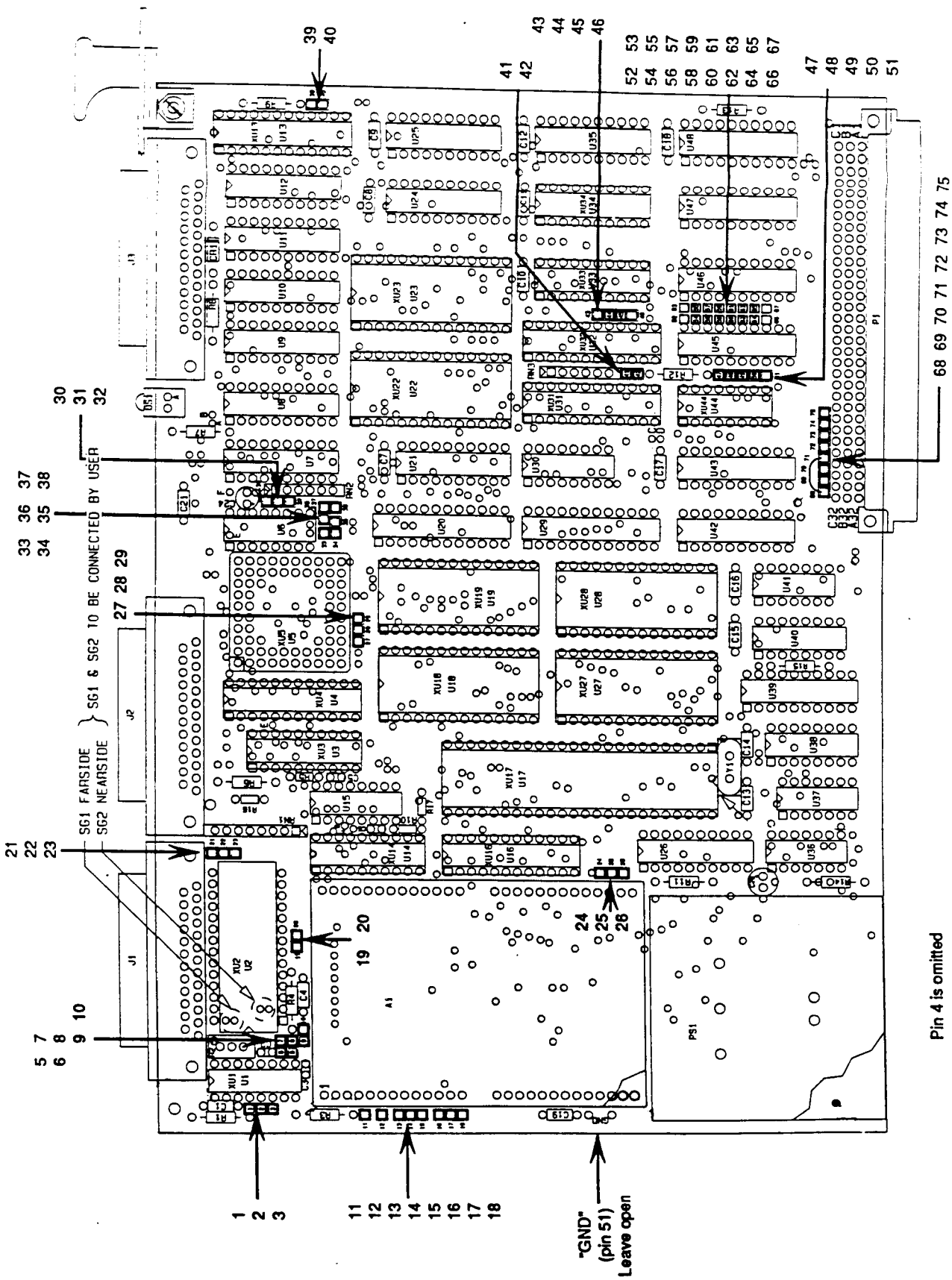


Figure 2.1 Assembly and Layout Diagram (Component Side) [Drawing D-15259]

2.1 Base Address Selection

In the following sections, please refer to the Assembly Diagram, Figure 2.1, to locate jumpers and other hardware. Position the board with the VMEbus connector, P1 at the lower right. Most of the factory jumpering will be correct for many applications. However, you should carefully check all jumpering before installing the board. Removable plugs are used for some connections and soldered jumpers or wire wrap posts are used for others.

The DVME-601 may be configured to occupy a 64 kilobyte block in the memory space of the host CPU. Jumper plugs are provided to select the BASE (lowest) address of the memory block using VMEbus address lines A23 through A16. Address lines A15 and below are locally decoded on the DVME-601 to address the DPR. Wire wrap connections may be used if preferred. The jumpers are located between integrated circuits U45 and U46 in the lower right just above P1. Please refer to Figure 2.2.

An installed jumper decodes a logic 0 for its associated address line. If the jumper is omitted, a logic 1 results. The DVME-601 is shipped at a standard BASE address of $FA0000 hexadecimal.

ADDRESS LINE	JUMPER
ADR 23	66 to 67
ADR 22	64 to 65
ADR 21	62 to 63
ADR 20	60 to 61
ADR 19	58 to 59
ADR 18	56 to 57
ADR 17	54 to 55
ADR 16	52 to 53

Jumper installed = 0, Jumper omitted = 1

Figure 2.2 BASE Address Jumpers

2.2 Address Modifier Selection

A jumper must be installed to select the desired Address Modifier code(s). The Address Modifier jumpers are located just to the left of the BASE address jumpers between U44 and U45. AM codes $29 and $2D are listed below but should not be used for the DVME-601. Codes $3A and $3E allow the host to fetch and execute instructions out of the DPR. Normally this mode would not be used but is possible if required. Note that the DVME-601 may always be accessed in Supervisory Mode. See Figure 2.3.

Jumper	Responds to	Access description
47-48	Code $39	Standard non-privileged data access
[STD.]	Code $3A	Standard non-privileged program access
	Code $3D	Standard supervisory data access
	Code $3E	Standard supervisory program access
47-49	Code $3D	Standard supervisory data access
	Code $3E	Standard supervisory program access
47-50	Code $29	Short non-privileged access
	Code $2D	Short supervisory access
47-51	Code $2D	Short supervisory access

Figure 2.3 Address Modifier Jumpers

2.3 Interrupt Level Selection

The DVME-601 may generate interrupts to the VMEbus host system using interrupt levels 1-7. There are several sets of jumpers as shown in Figure 2.4. The interrupt request jumpers are located immediately above the P1 VMEbus connector. Other interrupt jumpers are between U44 and U45 and between U32 and U33. These IC's are just above P1. Th desired interrupt level is selected by jumpers as follows:

Interrupt Level	Install Jumpers			
	BIRQ* Line Jumper	43-44	45-46	41-42
1	68 to 69	IN	IN	OUT
2	68 to 70	IN	OUT	IN
3	68 to 71	IN	OUT	OUT
4	68 to 72	OUT	IN	IN
5	68 to 73	OUT	IN	OUT
6	68 to 74	OUT	OUT	IN
7	68 to 75	OUT	OUT	OUT

Figure 2.4 VMEbus Interrupt Level Selection

2.4 A/D Converter Jumpering

The A/D Converter must be jumpered for single-ended or differential inputs, input voltage range, and output data coding. In addition, the AM-551 Instrumentation Amplifier, U1, may be bypassed. This offers slightly faster settling times and can only be used with single-ended inputs. Differential inputs always require the Instrumentation Amplifier.

Single-ended inputs use channels 0 through 15. Differential inputs use channels 0 through 7. Most jumpers are adjacent to the Data Acquisition Section in the upper left. Jumpers 39 and 40 are on the upper far right board edge.

2.4.1 Single-Ended/Differential Inputs

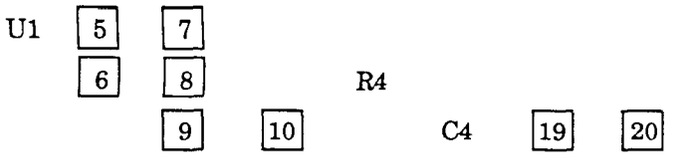
IMPORTANT! Both single-ended configurations should REMOVE jumper 22-23. Otherwise -15V dc will be connected to the channel address generator and damage could result. The differential configuration should connect ONLY jumper 22-23 as shown. Users should study the schematic drawing, sheet 5, to understand all the connections.

Single-ended input: (No inst. ampl.)
Install: 9-10, 19-20, 21-22
Remove: 1-2, 2-3, 5-6, 8-9, 39-40, 22-23

Single-ended input: (with inst. ampl.) [factory standard]
Install: 1-2, 5-7, 8-9, 19-20, 21-22
Remove: 2-3, 5-6, 9-10, 22-23, 39-40

Differential input: (with inst. ampl.)
Install: 2-3, 5-7, 8-9, 22-23, 39-40
Remove: 1-2, 5-6, 9-10, 19-20, 21-22

The jumpers near the AM-551 PGA (U1) are difficult to see on the board. They are as follows:



To use any of the expansion slave MUX boards (DVME-64X series), the DVME-601 must be configured for local differential input by carefully cutting the 19-20 jumper with miniature cutters or removing the 19-20 plug if it is installed. Local single-ended inputs on the 601 are not allowed if slave MUX boards are connected because the expansion bus is inherently differential. Single-ended input on the slave MUX board is allowed however.

2.4.2 Instrumentation Amplifier Gain Selection

The AM-551 Instrumentation Amplifier U1, is normally supplied with a gain of one. However the gain may be increased up to 1000 by changing a jumper and adding a gain resistor. The resistor is normally omitted and is installed in location R1 in the extreme upper left of the board.

There are several practical consequences of increasing the gain. The gain is related to the temperature coefficient of the gain resistor. Therefore R1 should be a high quality, low-TC metal film type. At higher gains, the AM-551 requires additional settling time and has somewhat lower small signal bandwidth. Refer to Figure 2.5. For values between those shown, the user should interpolate. Consult DATEL's AM-551 data sheet for more information.

Total Gain	Settling time, 20V output to 0.01%	Small signal bandwidth, -3 dB
1	2 µS, typ.	400 KHz typ.
10	4.6 µS, typ.	150 KHz typ.
100	20 µS, typ.	100 KHz typ.
1000	200 µS, typ.	40 KHz typ.

Figure 2.5 Instrumentation Amplifier Characteristics

Gain is selected for both the input and output stages of the AM- 551. The input stage may be continuously varied up to gain=100 by selecting R1. The value for R1 is determined by a formula. The output gain is either 1 or 10 by changing a jumper. The total AM- 551 gain is the input gain times the output gain.

Gain selection is accomplished in two stages. The input stage gain (G1) is selected by R1 and is expressed as follows:

$$R1 = 20 \text{ Kilohms}/(G1 - 1)$$

The output stage gain (G2) is selected by jumper. For G2 = 1, install jumper 5 to 7. For G2 = 10, install jumper 5 to 6.

$$\text{Total Gain} = G1 \times G2$$

A single gain resistor location is provided. Therefore, most standard value resistors may not provide exactly the gain required. To calibrate for higher gains, these are some possible alternatives:

1. The gain may be initially set with a precision standard value metal film resistor of slightly higher value (lower gain) than is required. The final gain is determined empirically with a voltage calibrator and a high-value trim resistor which parallels the precision resistor. The two resistors are soldered closely together and installed at the R1 location.

2. If a single standard value resistor provides gain that is very close to the desired amount, the gain control on the A/D converter module may be used to adjust the final value. Refer to the A/D calibration procedure since this may also alter the zero or offset calibration.

3. A single precision resistor may be used that provides approximately the correct value. Final gain is then done in software by multiplying all raw binary data values by a calibration constant. The arithmetic may be done locally on the DVME-601 before sending data to the host.

If two analog input channels are dedicated to ground and full scale reference inputs, the constant may be calibrated automatically and continuously with every Nth scan. Using both ground and full scale references, a first order Y = MX ±B equation will offer autozeroing as well. Thus each DVME-601 will be autocalibrating. There is of course a speed penalty for the software autocalibration but it may be combined with sensor linearization programming which was to be performed anyway.

Finally, you should be aware that total system gain must also consider the A/D converter input range, discussed in the next section. Using a lower A/D input range changes the bit weighting (the millivolt value of each output data count) just as effectively as changing the AM-551 gain.

2.4.3 A/D Input Range and Output Data Coding Selection

For many applications, the standard A/D input ranges available on the DVME-601 will be sufficient and extra gain will not be required in the Instrumentation Amplifier. The DVME-601A is supplied with up to four user-selected input ranges. Two ranges are unipolar and two are bipolar. The DVME-601E has a bipolar and unipolar range. The DVME-601B-U and -601C-U are fixed as unipolar and are available under special order.

Bipolar ranges use one data bit for polarity indication whereas unipolar ranges do not have a polarity bit. A 12-bit A/D converter may be considered either 12 unipolar data bits or 11 bipolar data bits and a sign bit.

A/D models and input range options are shown in Figure 2.6. Jumpers to select these ranges are shown in Figures 2.7 and 2.7a. Range jumpers are along the extreme left edge of the board outside the A/D converter module. A/D data output coding may be either unipolar straight binary, bipolar offset binary or bipolar two's complement.

A/D output coding is shown in Figure 2.8. Output coding jumpers are along the lower right corner of the A/D converter. The jumper inverts the converter's most significant data bit. Refer to the calibration section for information on coding and input ranges.

If possible, group all inputs together as sequential channels and avoid scanning through unused inputs. Scanning through open inputs will not delay scanning but may degrade settling performance on the next channel. If you must scan through unused inputs, consider grounding them or connect a high value input resistor to ground (say 50 Kilohms) on each unconnected input to prevent amplifier saturation. Unterminated inputs may display small stray MUX feed through and crosstalk signals.

Board Type	DVME-601A	DVME-601B		DVME-601C	DVME-601D	DVME-601E		DVME-601F
A/D Module Type	ADC-12/20	ADC-12/2	ADC-12/2A	ADC-16/35	ADC-16/400K	ADC-12/2	ADC-14/2A	ADC-14/2
Full Scale Input Ranges	Jumper Selections							
0 to +5V	16-17, 13-18 14-15	N/A	N/A	N/A	N/A	N/A	N/A	N/A
0 to +10V	16-17, 14-15	16-17	16-17	(-U model) 16-17, 13-14	N/A	16-17	16-17	E2-E3
±5V	16-17, 13-14	N/A	16-17, 13-14	16-17, 13-14	N/A	N/A	16-17, 13-14	E1-E2, E4-E5
±10V	17-18, 13-14	13-14	17-18, 13-14	17-18, 13-14	17-18, 13-14	13-14	17-18, 13-14	E1-E2

*Note: DVME-601F models are supplied and specified as ±10 Volt full scale inputs but may be jumpered on the ADC-14/2 module only (not the 601 board) for 0 - 10V or ±5V. See the ADC-14/2 module drawing for details. On the DVME-601F jumpers, E1 - E5 are on the bottom surface of the A/D module. You must remove the A/D module to change the setting. Recent DVME-601B and -601E boards use ADC-12/2A modules.

Figure 2.6 A/D Input Ranges and Model Types

A/D Output Data Coding	Polarity	Install Jumper
Straight or Offset Binary	Unipolar	24 to 25
Two's Complement	Bipolar	25 to 26

Figure 2.7 A/D Output Data Coding

2.4.4 DVME-601F Range Jumpers

Range -1	Internal Jumper -2	External DVME-601F Board Jumpers
0 to +10V	E2-E3	Leave all board jumpers 17-21 open
±5V	E1-E2 E4-E5	Leave all board jumpers 17-21 open
±10V (Standard)	E1-E2	Leave all board jumpers 17-21 open

Notes:

1. The 0 to 10V and ±5V ranges may experience increased noise. DATEL suggests averaging a sum of adjacent multiple data samples while locked on one channel.

2. The "E" jumpers are on the ADC-14/2 analog module. Unlike other -601 modules, range adjustments must be made on the module. To avoid damage, range jumpering should only be performed by a skilled person.

3. Differential mode is strongly recommended for 14-bit A/D operation.

4. The averaging technique in Note 1 may be performed very quickly in assembly language. First, reset the autoincrement bit in the command register (CMD3 = 0). Write the channel address, allow 6 microseconds minimum settling then start A/D conversions. Sum the data into the CPU accumulator, collecting a power-of-two number of samples. While locked on one channel, no settling time is required after the first conversion. Data may be read as fast as the EOC appears and fast throughput mode (CMD5 = 1) may be used to eliminate EOC polling. Now perform a root 2 right shift of the summed data, or simply use the larger sum.

   Example: Sum 8 A/D samples and shift right 3 bits (LOG2 of 8=3). This avoids slow floating point averaging and can all be done in assembly language.

2.5 Auxiliary RS-232-C Serial Port Jumpers

The J2 front panel connector serves several purposes. It is the primary connector for the 68901 peripheral controller but also accepts an external A/D trigger start input. The 68901 controller provides 5 TTL I/O bits, 2 timer/counter inputs, 3 timebase outputs and an RS-232-C serial port. This port is full duplex with synchronous and asynchronous character formats. It is pinned out as RS-232-C Data Terminal Equipment (DTE) as if it were a terminal. The discussion below assumes an understanding of RS- 232-C specifications and serial data transmission. Please refer to the J2 connector pinout discussed in Section 3.

The serial port is the primary interface for the firmware Monitor. When connected to an external dumb ASCII terminal, the Monitor is used to develop user programs, for A/D calibration and for diagnostics. After the Monitor is used, the J2 serial port may be reassigned by local user programs to other functions. When used as a serial port, the J2 connector and the 68901 controller offer a subset of the full RS-232-C interface. Most of the RS-232-C handshake lines are available depending on jumpering described below. The other 68901 controller connections on J2 for timer/counter/I-O are not part of the RS-232-C specification and should be reviewed before making connections to your Monitor terminal. This may be accommodated by wiring a special cable. Refer to Section 3 covering I/O connections.

Five of the general purpose I/O lines of the MC68901 controller are brought out on the J2 connnector as single load TTL logic levels. Two of these I/O bits, 0 and 1 on pins 21 and 22 may also be jumpered on the board to an unused RS-232-C line driver and to one of two unused RS-232-C receivers. These spare RS-232-C devices connect to J2 in the proper locations to act as RS-232-C handshakes for the 68901 serial port. They may also be programmed as a second serial port using a software UART.

Jumpering for the spare RS-232-C output is shown in Figure 2.9. The two spare RS-232-C inputs are jumpered according to Figure 2.10.

When these jumper connections are made, the raw TTL levels for I/O bits 0 and 1 will still be available on the J2 connector in addition to the RS-232-C levels. Be careful if you make connection to the I/O bits to preserve operation of the RS-232-C devices. Please refer to the DVME-601 Schematic Diagram, drawing number D-15260, sheet 4.

Jumper	RTS Output (J2 pin 4)
31 - 32	Always true if power is on.
30 - 31 (use for aux. ser. output)	Programmed by I/O bit 1 as output.

Figure 2.9 Spare RS-232-C Output

Jumper	Input source to I/O bit 0
37 - 38	From CTS, J2 pin 5
36 - 37	From DSR, J2 pin 6

Figure 2.10 Spare RS-232-C Inputs

2.5.1 RS-232-C Serial Port Handshakes

The four standard RS-232-C handshakes (CTS, DSR, DTR, and RTS) connect to transceivers on the DVME-601. The Data Terminal Ready output (DTR) is always true at J2 pin 20 when power is on. The Request To Send output (RTS) at J2 pin 4 may be jumpered always true or may be software programmed by output bit 1. Most terminals connected to J2 will require these lines to be asserted.

Either (but not both) of the Clear To Send (CTS) or Data Set Ready (DSR) inputs at J2 pins 5 and 6 respectively may be jumpered to bit 0. User software would program this bit as an input. The 68901 controller has no internal CTS or DSR handshake logic to stop transmission when an external DCE device requests a halt. Using them as data flow control handshakes would require polling I/O bit 0 by a user program. Bit 0 may also be programmed as a local interrupt to avoid polling.

2.5.2 RS-232-C Auxiliary Software Serial Port

The RS-232-C spare transceivers may also be programmed as a second software serial port. Serial input data is connected to either J2 pin 5 or 6. Software UART routines in the firmware use the auxiliary serial input as an S2-record downloader. The input defaults to 4800 baud, 8 data, no parity, 1 stop bit, asynchronous format.

The input allows the user to download S2-record programs from the host or an external development system. This is an alternative software channel to downloading through the DPR. It has the advantage that it is immediately compatible with most serial-loading development systems and does not require the user to write a DPR downloader right away. When using the downloader, both the controlling Monitor terminal and the download serial input must be connected. There is no hardware handshaking on the auxiliary serial port unless the user wanted to program the auxiliary RS-232-C output.

The RS-232-C output may be user-programmed by controlling I/O bit 1 on J2 pin 4. To form a serial output, a software output UART subroutine is provided in EPROM. The local program can call the subroutine.

2.6 Analog and Digital Ground Connection

The analog and digital grounds are connected together inside the S/H-A/D module where they produce the highest data quality. The digital/power ground is also connected to the host system ground through the VMEbus backplane ground. The host may ultimately connect power ground to earth (chassis) ground through the AC third prong. Or the chassis may remain fully transformer isolated from earth ground. You should fully understand all implications of these ground connections to avoid large ground loop currents flowing through your input transducers or possible damage to the DVME-601.

In your complete application circuit, there must one and only one connection between the analog and digital/power ground systems (i.e., the connection inside the module). Using two or more parallel ground connections may introduce ground loop errors. The mistake commonly made is to ground the sensor shields to earth ground remotely out at the sensor while the shields are also connected to analog ground. This creates a large, single-turn transformer called a ground loop. The amount of noise current collected by this loop is proportional to the noise fields and the loop area. A fundamental rule is that ground shields should carry no current (including induced ground loop current).

CAUTION: The DVME-601 on-board channels are not isolated and can be damaged by excessive common mode voltages. If your application requires galvanic isolation between the two ground systems, consider using the DVME-643 slave isolated MUX board to connect all inputs. Contact DATEL if you need assistance.

2.7 EPROM Memory Address Jumpers

The DVME-601 is supplied with 64 kilobytes of EPROM memory containing the Monitor and Executive firmware. This is provided in two type 27C256 EPROM's at U18 and U19 arranged in 16-bit word addressing. Please refer to the schematic drawing, D-15260, sheet 3.

Users with special program requirements may wish to consider larger EPROM's up to 128 Kb maximum. To configure 128 Kb of memory, address jumpers must be changed and new EPROM's will be required. Two type 27C512 devices should be installed at U18 and U19. The new EPROM's must be programmed properly before use. The installation should only be considered by a user skilled at electronic assembly. The reprogramming is discussed in a later section.

The DVME-601 will also accept lower cost 2764 and 27C128 EPROM's if required. Jumpering for all EPROM sizes is shown in Figure 2.11 below. The jumpers are adjacent to the 68010 microprocessor U5 in the top center of the board. Alternate EPROM sizes are available from DATEL under special order.

Total EPROM memory	U18 & U19 device types	Install jumpers
16 Kb	(2) 2764/C64	34-35, 27-28
32 Kb	(2) 27C128	34-35, 27-28
64 Kb	(2) 27C256 [supplied]	33-34, 27-28
128 Kb	(2) 27C512	33-34, 28-29

Figure 2.11 EPROM Address Jumpers

2.8 Board Installation

All DVME-601 jumpers should now be properly installed. Confirm that your host and its operating system function normally. Shut off host power.

Consult your host documentation to determine the proper board slot. The DVME-601 preserves the VMEbus Interrupt Acknowledge and Bus Grant chain. The board should be positioned in the next free board slot away from the host CPU board. If you must leave empty slots between the DVME-601 and the other system boards, be sure backplane IACK jumpers are in place for the empty slots. IACK jumpers should be out for the slot used by the DVME-601 if you want interrupts. Rearrange the sequence of boards if you want to vary the hardware priority of DVME-601 interrupts. Higher priority boards are nearer to the CPU board. The DVME-601 also includes board connections to daisy-chain bus grant signals used by DMA controllers.

Install the board carefully using anti-static protection. Do not force the board. If it does not seat properly, remove the board and determine the cause. Observe the P1 connector if possible to see that it is seating properly.

Now reapply power with the DVME-601 installed. Confirm that your operating system boots up normally. Verify the memory size message if it is normally present. The DVME-601 front panel green LED lamp should light to confirm power-up self testing of its CPU, memory and peripheral controller. If your operating system fails to boot up normally, turn power off immediately and review the board configuration information in this manual.

Do not make any connections to the front panel until you have read this manual. Avoid any live analog connections with power off.

I/O CONNECTIONS AND THE MONITOR

3.0 Introduction

The DVME-601 includes three front panel connectors and a P1 VMEbus connector. The front panel connectors are female 25-pin type DB-25S. They include 4-40 threaded strain reliefs for optional hooded mating DB-25P connectors. The rear VMEbus connector is a male 96-pin 3-row DIN type. A programmable Light Emitting Diode (LED) green lamp is also mounted on the front panel. The layout for the front panel is shown in Figure 3.0.

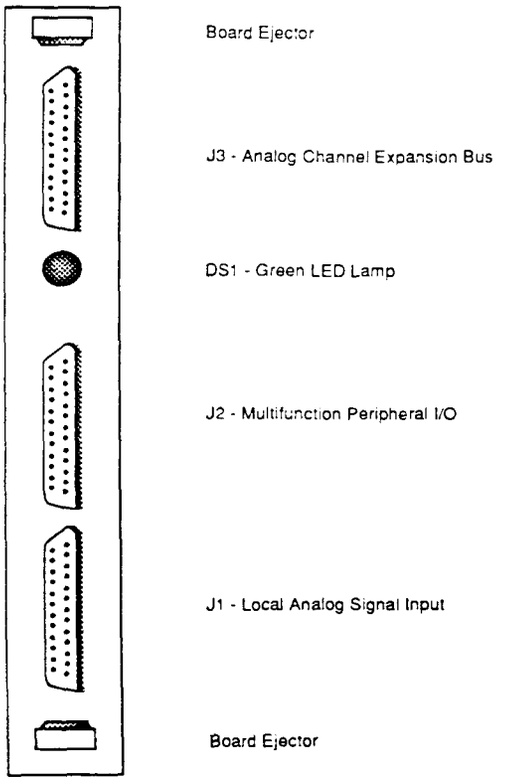


Figure 3.0 Front Panel Connector Locations

3.1 Local Analog Signal Inputs

The J1 connector accepts voltage signals for on-board analog input channels. The connector pinout is arranged to accept either single-ended or differential inputs depending on the board jumpers. For single-ended inputs (Figure 3.1), channel numbering is 0 through 15. Differential inputs (Figure 3.2) are channels 0 through 7. Both figures show the pin numbering as viewed looking toward the front panel.

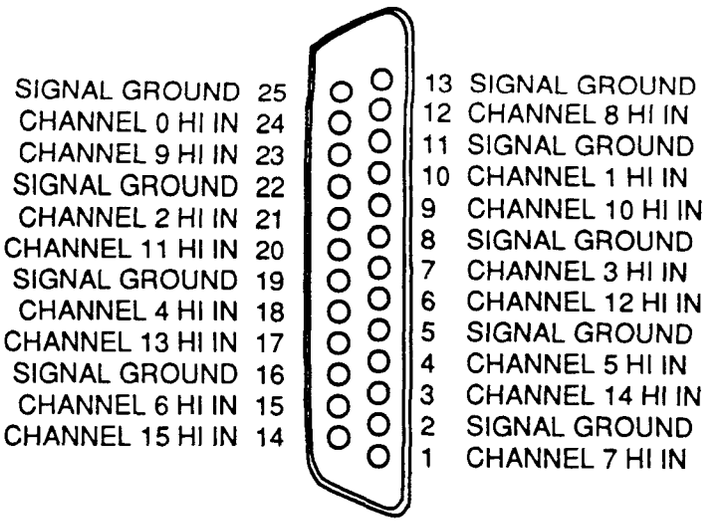


Figure 3.1 Local Single-ended Analog Inputs - Connector J1

The grounds are arranged so that a ground is shared by every two channels.

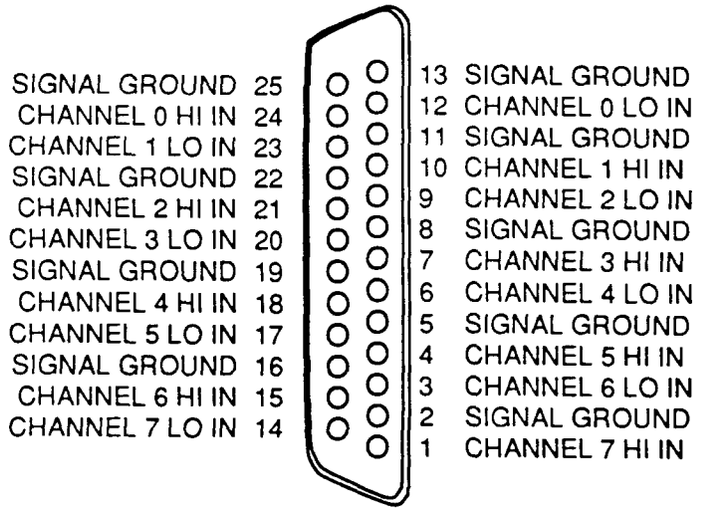


Figure 3.2 Local Differential Analog Inputs - Connector J1

Pin numbering is arranged as viewed looking toward the front panel.

3.1.1 Typical Analog Input Wiring

Figures 3.3 and 3.4 show typical input wiring used with both single-ended and differential input sources. Note that single-ended inputs share one ground for approximately every two input channels. The following guidelines should be observed:

1. All inputs are non-isolated. All inputs must remain within the common mode voltage range at all times during measurement. This is accomplished in your application by providing a bias current return path to ground through the sensor or bridge.

If you need fully isolated, floating inputs to the DVME-601, consider connecting it to the DVME-643 slave multiplexer board.

2. For the single-ended inputs, two external connections (Hi and signal ground) are required per channel. On differential inputs, three connections are required per channel (Hi, Lo, Ground). Do not omit the ground connection or low input on differential inputs.

3. Use a single point system analog ground connection. Multiple ground paths form ground loops which collect noise which is proportional to the loop area and noise field strength.

4. If you use shielded cable, avoid carrying any current on the shield. This will prevent noise induced on the signal leads. One way to do this is to terminate the shield only at the grounded end. If so, you will still need a bias return path for the bridge ground. See Figure 3.5.

5. Avoid high amplitude wideband noise on input signals. The narrow aperture and fast acquisition time of the DVME-601 sample/hold amplifier and A/D converter will faithfully digitize the noise as well as the signal. Large amplitude noise may also be partially rectified in input circuits, producing dc offset errors. Consider external filtering for high noise inputs.

6. Fully attenuate all input signal frequency spectra which are at or above one half the sample rate of the converter. This will avoid alias noise. Use multipole filters if necessary.

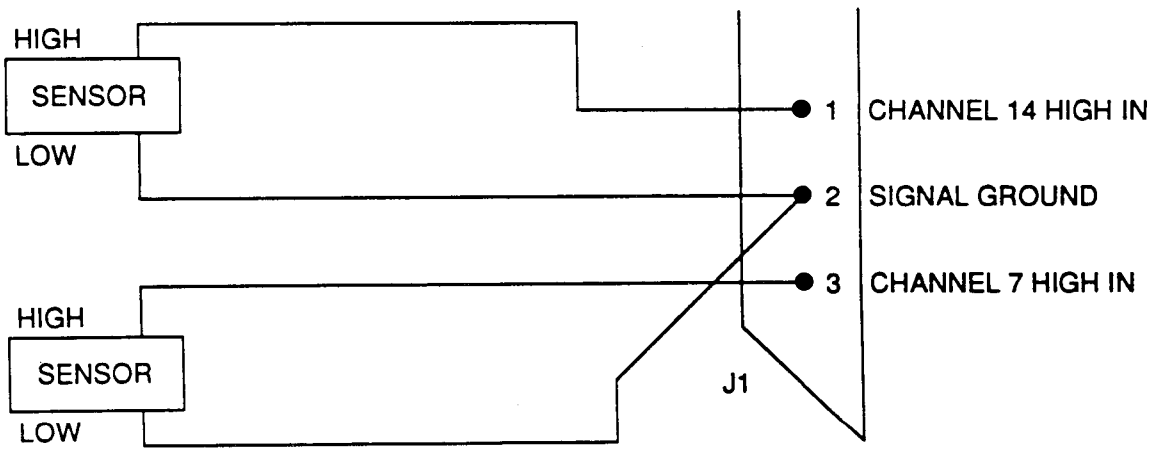


Figure 3.3 Single-ended Input Wiring

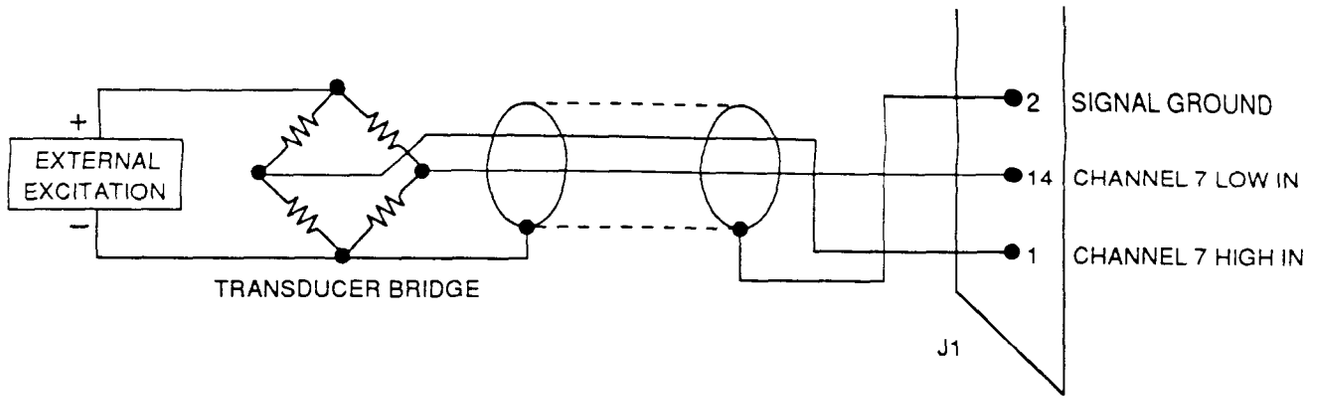


Figure 3.4 Differential Input Wiring

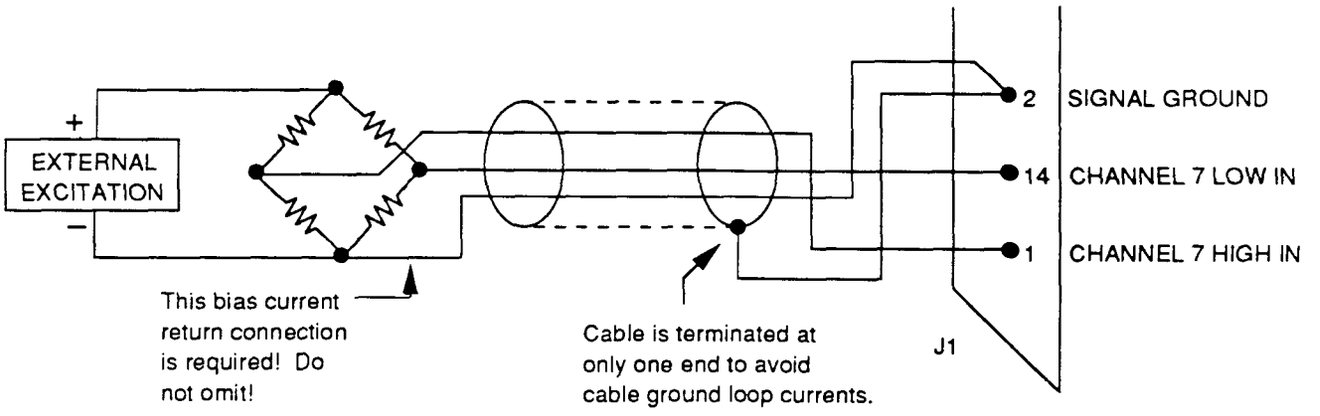


Figure 3.5 Differential Bridge Wiring

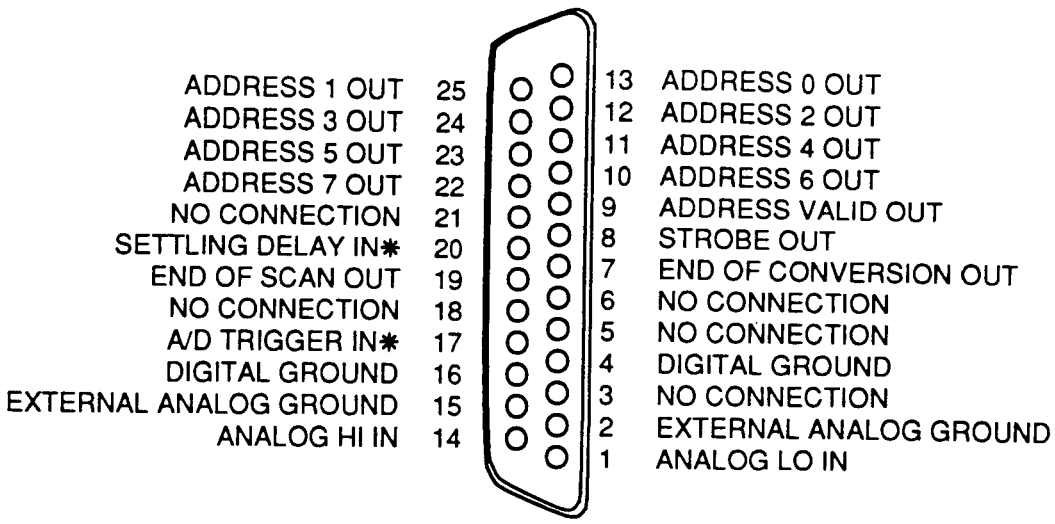
3.2 A/D Channel Expansion Bus

The DVME-601 offers channel expansion for additional inputs beyond the local on-board channels. The maximun limit is 10 slave MUX boards. The DVME-601 local channels must be configured for differential inputs before using channel expansion. A flat cable channel expansion bus is provided on the DVME-601's J3 front panel connector. It plugs into DATEL's optional slave multiplexer boards installed in slots adjacent to the DVME-601 or in a nearby VME chassis. The pinout of the J3 connector is shown in Figure 3.6. An overall view of front panel connections including the channel expansion bus is shown in Figure 3.7.

NOTICE

Channel expansion is normally disconnected:

Starting with PC board artwork Revision F, the expansion analog inputs are supplied disconnected. Connect solder gaps SG1 and SG2 on the non-component side of the board to enable expansion inputs. Use a low wattage soldering iron to avoid damaging the board. This operation should only be done by an experienced person. If you anticipate any difficulty in doing this, contact DATEL before shipment.



Pin numbering is arranged as viewed looking toward the front panel.

Figure 3.6 Analog Channel Expansion Bus - Connector J3

3.2.1 Channel Expansion Multiplexer Boards

The slave MUX boards include the DVME-641 32S/16D channel high speed MUX, the DVME-643 8D channel low level isolated MUX (for sensors such as thermocouples, RTD's, 4-20 mA loops, etc.) and the DVME-645 16S/8D channel simultaneous sample/hold MUX. The DVME-645 is especially suited to array processsing and DSP applications.

The MUX boards use +5V dc power from the VMEbus to run on-board dc/dc power converters. This generates power for linear circuits. Otherwise the MUX boards do not connect to the VMEbus.

Channel address programming is discussed in Section 4. The channel expansion bus allows the DVME-601 to directly control each slave MUX board and carries three classes of signals. They are:

1. The bus includes 8-bit channel address outputs from the DVME- 601's current channel address register. The eight address outputs are J3 pins 10 through 13 and 22 through 25.

2. Buffered high level switched differential analog input signals are supplied on pins J3-1 and J3-14. These inputs use J3-2 and J3-15 as grounds. A local multiplexer on the DVME-601 switches these inputs into the Instrumentation Amplifier for A/D conversion.

3. Also included are control and handshake lines, an external A/D start trigger input and grounds.

Channel addresses are distributed to all MUX boards along the bus. The 8-bit binary address code controls up to 256 total single-ended or differential channels. The channel addresses are numbered 0 through 255 and the first 8 differential or 16 single-ended channels are always located on the DVME-601.

Individual multiplexer boards include base address decoder switches which must be set properly before installation. Address selection logic allows each MUX board to respond to a unique range of addresses. When selected, a MUX board connects its analog outputs to the differential input in pins J3-1 and 14. These inputs are digitized by the DVME-601's A/D converter. All other de-selected MUX boards disconnect their analog outputs from the bus until they are addresssed. This analog multiplexing occurs at very high speed. There is no speed penalty addressing a range of contiguous channels over several MUX boards. Nor do contiguously addressed MUX boards have to be adjacent to each other or even in the same VME rack housing.

Both single-ended and differential inputs may be mixed on different MUX boards for the same bus while local DVME-601 channels must remain differential. For diagnostics, a LED lamp on each MUX board lights when that board is addressed.

DATEL offers 2- and 3-connector cables, models DVME-C-01 and -02, to connect one or two slave MUX boards and users may fabricate flat cables for up to 10 boards total.

Pin J3-17 on the expansion bus is a TTL, open-collector, negative-true A/D trigger input. The A/D start input to the DVME- 601 may be initiated from the trigger input on any multiplexer board. A pullup resistor to +5V is supplied on the DVME-601. A single TTL hardware trigger from an external event will start either one A/D sample and host interrupt or a scan of channels on one or more boards. The trigger must be 100 nanoseconds minimum to 2 microseconds maximum. Triggering occurs on the falling edge.

Important: Disconnect the expansion trigger input on J3-17 if the local Pacer trigger connection (J2-19 to J2-10) is used.

Scan addressing is fully controlled from the DVME-601's start and final channel address registers. Scan address boundaries are independent of MUX board address boundaries. Thus one MUX board trigger input may start conversion for channels on other MUX boards.

Alternatively, automatic channel sequencing and A/D conversion may be started from a DVME-601 Pacer timer in the 68901 controller. The timer output must be jumpered to the trigger input on the J3 connector. Or a software command from the host will start the A/D.

An open collector wire-OR'd settling delay control line input is provided on pin J3-20. A settling delay one-shot output from each MUX board will delay the actual A/D conversion to synchronize settling times of low level preamplifiers on the DVME-643 slave MUX boards. High level DVME-641 and -645 MUX boards do not use this settling delay. If noise on this settling input causes random A/D delays, this line may be diconnected at the A/D board. Through appropriate host A/D software, board addressing and input range selection, it is even possible to mix high and low level MUX input boards on the same bus.

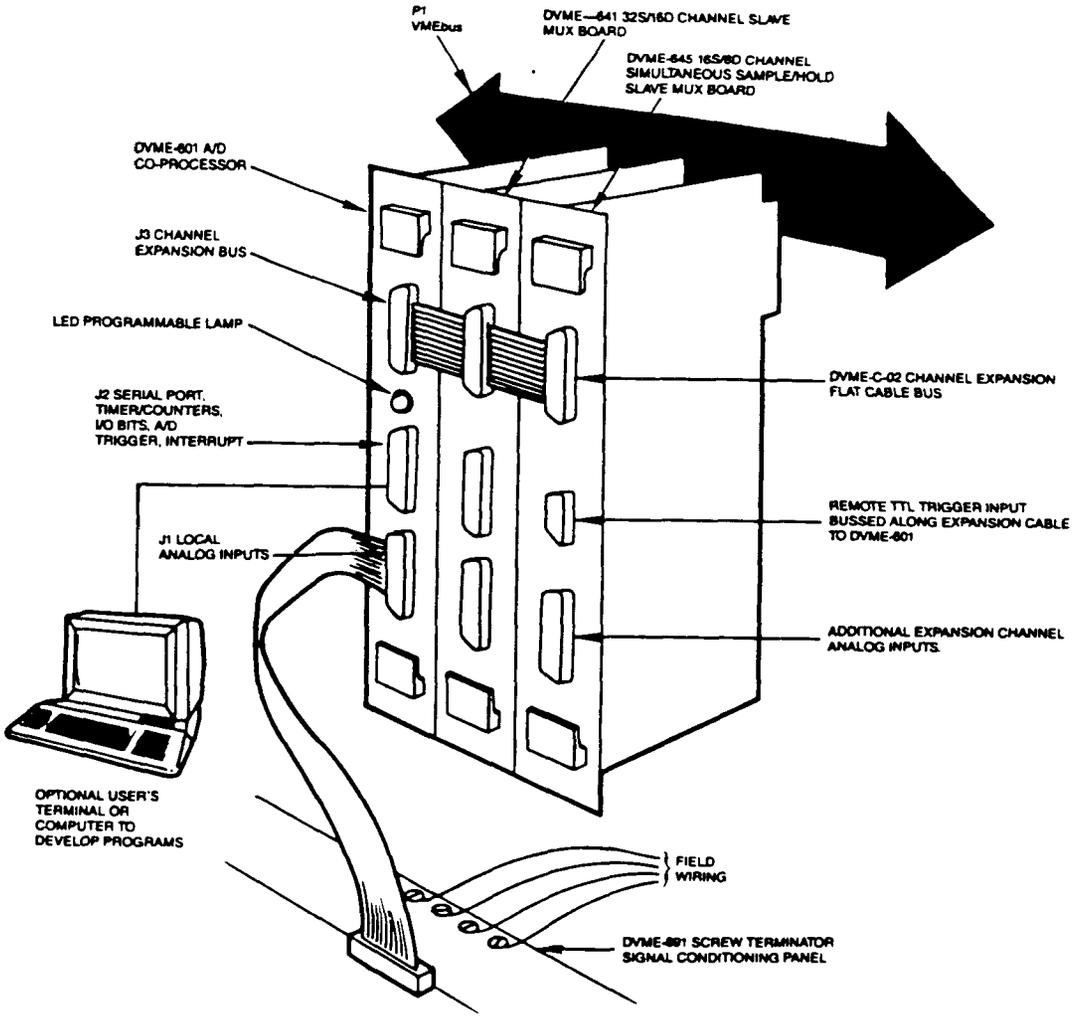


Figure 3.7 Front Panel Cabling and Channel Expansion

3.3 VMEbus Connections

The interface to the VME host is made through the P1 connector, shown in Figure 3.8. This interface uses the IEEE-P1014/D1.0 specification. Note that the Interrupt Acknowledge connections are fully implemented and that internal DVME-601 logic uses the 16 MHz SYSCLK. The ±12 Volt power pins are used if the serial port Monitor terminal is connected.

Pin Number	Row A Signal Mnemonic	Row B Signal Mnemonic	Row C Signal Mnemonic
1	D00	BBSY*	D08
2	D01	BCLR*	D09
3	D02	ACFAIL*	D10
4	D03	BG0IN*	D11
5	D04	BG0OUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	GND	BG2OUT*	GND
10	SYSCLK	BG3IN*	SYSFAIL*
11	GND	BG3OUT*	BEAR*
12	DS1*	BR0*	SYSRESET*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM5
15	GND	BR3*	A23
16	DTACK*	AM0	A22
17	GND	AM1	A21
18	AS*	AM2	A20
19	GND	AM3	A19
20	IACK*	GND	A18
21	IACKIN*	SERCLK	A17
22	IACKOUT*	SERDAT	A16
23	AM4	GND	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31	-12V	+5V STDBY	+12V
32	+5V	+5V	+5V

Figure 3.8 VMEbus Connector - P1

3.4 Peripheral I/O Connections

Figure 3.9 shows the J2 I/O connections used with the 68901 controller. For now, these connections will be used for the Monitor terminal described in Section 3.5. After you have written A/D programs in your host, you may wish to use the other I/O peripherals on J2. Jumpering for J2 was described in Section 2.

CAUTION - J2 includes both RS-232-C and peripheral I/O connections. Be sure pin connections do not conflict with your terminal. The RS-232-C pins are wired as DTE. A crossover RS- 232-C cable will be required for most DTE terminals and small computers.

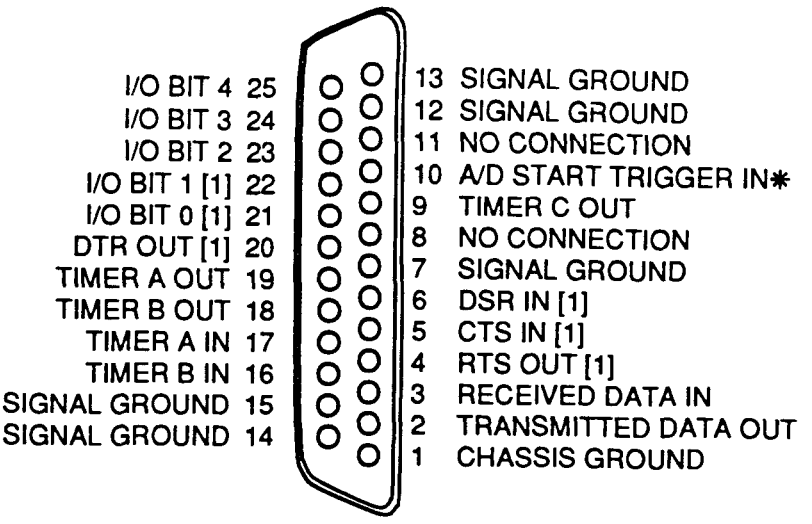


Figure 3.9 Multifunction Peripheral I/O - Connector J2

Note [1] Refer to Section 2 to implement internal jumpers and the RS-232-C handshakes on pins 4, 5, 6 and 20. DTR Out (pin 20) is always true when the board is powered up. RTS Out (pin 4) may be jumpered always true or programmed by jumpering to I/O Bit 1 (pin 22). Either CTS or DSR In (pins 5 and 6) may be read when jumpered to I/O Bit 0 (pin 21). Pins 2 - 6 and 20 are standard RS-232-C levels when implemented. I/O Bits are 1 TTL load each. Pin numbering is arranged as viewed looking toward the front panel.

3.5 Monitor Terminal Connections

If you have followed the sequence in Section 2, the DVME-601 has been properly configured and installed. You have powered up the host and verified that the host operates normally with the DVME- 601 installed. The DVME-601 LED lamp should be lit, confirming that the board has successfully passed power up self-testing. The board will start running the Monitor firmware, described below. The Executive portion of the firmware will be off, meaning that the board will ignore DPR activity by the host.

Using the built-in firmware Monitor with an external dumb ASCII terminal with is highly recommended for initial board familiarization. You will be able to send A/D data out to the terminal, perform calibration, download and execute programs. The VMEbus will only be used to supply dc power for these initial tests.

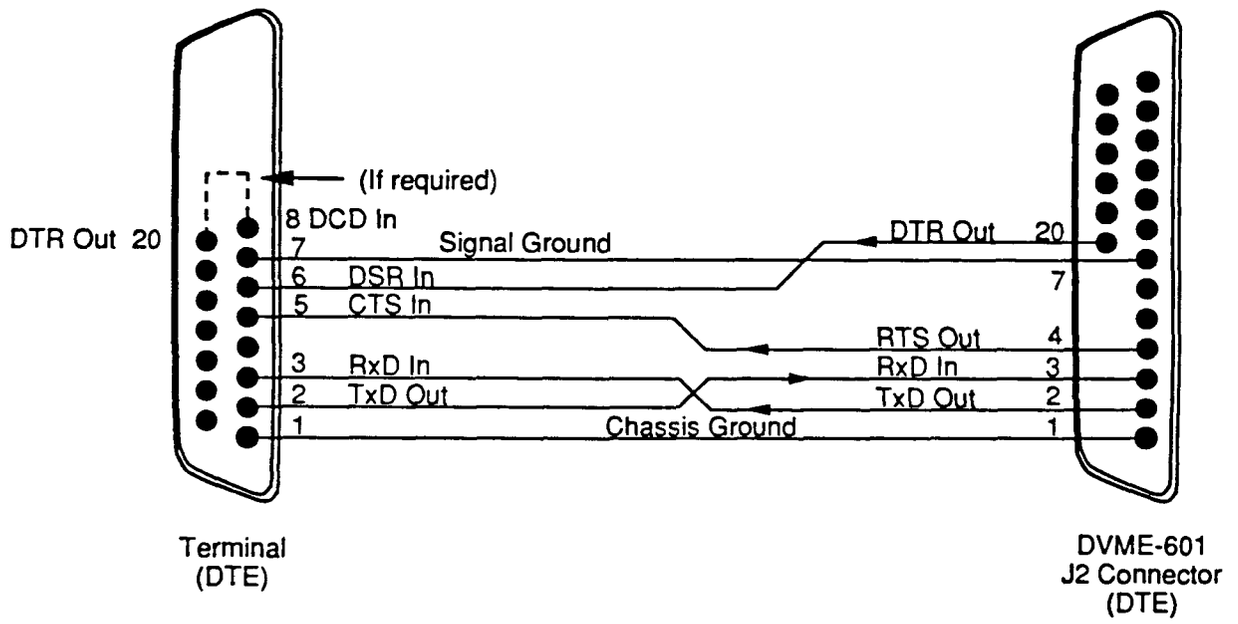
3.5.1 RS-232-C Terminal Cable

The next step is to fabricate an RS-232-C cable and configure your terminal. Normal RS-232-C interfacing connects a device with a Data Terminal Equipment pinout (DTE) to a Data Communications Equipment (DCE) pinout device. A terminal-to-modem connection is typical. When this is done, line names will be meaningful at both ends of the cable. Any one line will have only one line name.

Since your terminal and the DVME-601 are both DTE pinouts, a crossover cable must be used. In addition, line names will be crossed but the interface will function properly. We must also avoid connection to the non-RS-232-C functions on J2. These might interfere with operation of your terminal.

Most dumb terminals require their CTS and DSR inputs to be asserted at all times. These will be crossed over to the RTS and DTR outputs on the DVME-601. Some terminals may also require their Data Carrier Detect input (pin 8 on the terminal's DB-25 connector) to be asserted. Check your terminal manual to properly set the configuration switches. If you have configured the board with jumper 31-32 installed, both RTS Out (pin J2-4) and DTR Out (pin J2-20) on the DVME-601 will always be true. The DVME-601 does not require the CTS and DTR inputs (pins J2-5 and J2-6) to be asserted. You may also be able to simply jumper pins 4 to 5, 20 to 8 and 20 to 6 on your terminal connector to avoid handshake wiring from the DVME-601. In that case, you would only need the transmit and receive data lines and the grounds.

The RS-232-C cable between the terminal and the DVME-601 is shown in Figure 3.10.



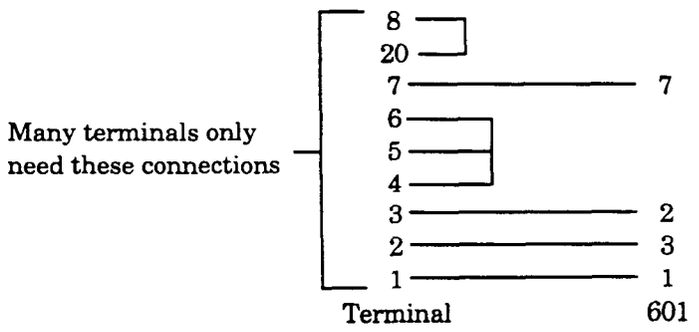


Figure 3.10 Monitor Terminal Cable Wiring

The terminal configuration switches must also set the baud rate and other parameters according to Figure 3.11. Some terminals allow these parameters to be set from a configuration menu on the terminal display rather than from switches. The parameters are then stored in non-volatile memory. The terminal width should be at least 80 columns wide.

Baud rate	9600 baud
Protocol	Asynchronous full duplex
Character Format	1 start bit, 8 data bits, no parity bit, 1 stop bit
Data type	ASCII per ANSI X3.4-1977 (All data characters are echoed as upper case to the terminal).
Number of columns	80 characters minimum

Figure 3.11 Monitor Terminal Parameters

3.5.2 Monitor Control Characters

Figure 3.12 lists important control characters which your terminal must be able to send. Control character assignment by the Monitor is conventional ASCII usage. Your terminal must respond to the Backspace control character exactly as shown. The Monitor echoes (transmits back to the terminal display) data and control characters which most popular terminals will support.

Smart terminals with cursor positioning and other attributes normally support these dumb terminal functions as a subset. Control characters are sent by holding down the control key on your terminal then pressing the appropriate alphanumeric key. The carriage return key may be labelled "CR", "RETURN", "Enter" or simply a left arrow on your terminal. The Monitor ignores the eighth bit of any character sent. Hexadecimal codes for each of the control characters are shown in parentheses. A line feed sent to the display with its cursor in the bottom line should scroll the display. Characters beyond 80 columns should wrap to the next line and cause a scroll if the wrap is beyond the bottom line.

ASCII control characters received by DVME-601:	DVME-601 internal action:
Backspace control H (08h) (Echoes as Backspace-Space-Backspace)	Removes last character form command buffer. Backspace must cause your terminal to move its cursor backwards. The next received character must overwrite the cursor location.
Delete (7Fh)	Same as Backspace
Carriage return (0Dh) ctl-M (Echoed as the prompt string <cr><lf>MON> after execution.)	Causes execution of previous command. Issues new prompt if successful. Sends Bell (07h) and '?' (3Fh) then new prompt if command fails.
ESCape control -[(1Bh) (ESC is not echoed to the terminal)	Clears command buffer then issues '$' prompt. On-screen characters are not erased.
Control-S [XOFF] (13h) (XOFF is not echoed to the terminal)	Temporarily suspends data transmission from DVME-601. Used during long memory or A/D display.
Control-Q [XON] (11h) (XON is not echoed to the terminal)	Clears XOFF and restarts transmission.
Control-C [ETX] (03h) (Ctrl-C is not echoed)	Transitions back to Monitor if in the Exec mode. See description below.
Control-X [CAN] (1Bh) (Ctrl-X is not echoed)	Performs hard reset to Monitor power-up state. Recovers CPU lockups.
Other control characters not listed	Ignored and echoed as a pound '#' sign.

Figure 3.12 Monitor Terminal Control Characters

3.6 Monitor Commands

The DVME-601 EPROM firmware contains a conventional microcomputer Monitor program. It is similar to Motorola's DEbug Monitor supplied with the VERSAdos Operating System but has unique DVME-601 features. The Monitor allows you to read and write local memory, the DPR or CPU registers, and execute programs under breakpoint or trace control. An S2-record program downloader is included for the auxiliary serial port. The Monitor also includes an A/D conversion command to aid in troubleshooting or diagnostics.

3.6.1 Power Up Activity

When the DVME-601 is first powered up, it does a checksum test of the EPROM, does a read/write test of RAM and tests the 68901 controller. This takes about one second. Selftest does not check the DPR in case the host has temporarily used it. (The DPR may be tested with the Monitor BF and MD commands).

The power-up software sets three bits for these tests in an internal EXCSTAT system variable. If all three items test okay, the DVME-601 does further system initialization and lights the LED lamp. The DVME-601 then runs the Monitor and sends the Monitor prompt out the serial port to your terminal. It will attempt to run the Monitor and send the prompt even if other power-up selftests fail and the LED

lamp remains off. The prompt message is a carriage return and line feed control characters then the characters 'MON>' as follows:

cr lf MON>

Normally the terminal's cursor will be positioned directly after the prompt message. The Monitor will now wait for you to enter commands. Upper or lower case command letters are accepted but will echo them to your terminal as upper case. If you make a mistake, use the Backspace (BS) or Delete (DEL) control characters. Each BS or DEL will erase one character and move the cursor backwards up until the prompt. The last character sent should be a carriage return which will start execution of the command you have typed.

3.6.2 Notation and Syntax

In the Monitor command listing in Section 3.6.3, the following notation is used:

<hex> Angle brackets indicate variable parameters entered by the user. If these parameters are not enclosed by square brackets, the parameters are REQUIRED. All data and addresses must be entered as hexadecimal. Leading zeroes may be omitted. Excess digits will be truncated or may abort the command. Addresses are relative to the local CPU, not the host.

<reg> CPU registers are designated D0, D1, ... A0, A1, ... and PC, ... (See the RM command).

[<opt.>] Square brackets indicate optional parameters. They must be separated from previous parameters by a delimiter. A space or comma ',' delimiter may be used except that count or range items require a comma.

(cr) All Monitor commands are terminated with a carriage return control character which executes the command. The (cr) is not shown in the command listing below.

Once the command is entered, <cr> will also enter new data for memory or register modify commands.

dot<cr> A single period and carriage return ".<cr>" will terminate several commands such MM, RI and RM. A new prompt will result.

BS/DEL Once a command has begun execution and expects further parameters (such as modify commands), BS or DEL will allow the user to correct bad entries.

Refer to Figure 3.12 for important control character usage with the Monitor. In particular, XOFF/XON will be helpful to pause output to the display.

Important: All numeric parameters must be in hexadecimal.

3.6.3 Monitor Command Listing

Commands used by the Monitor in this section have the format:

COMMAND

Description and function.

Monitor Commands

H

This displays a HELP menu and syntax of all commands.

RD

Display all 68010 internal CPU registers.

RM <reg>

Display/Modify CPU registers, where <reg> can be:

D0 -> D7 for data registers.
A0 -> A7 for address registers.

PC - program counter. SR - status register.
USP - user stack pointer. SSP - supervisor stack pointer.
VBR - vector base register.
SFC and DFC - alternate function code registers.
(SFC and DFC are display-only, no modify).

Depending on which register is used to invoke the command, the registers will cycle around within each of the two register groups above. Use ".<cr>" to exit.

RI [<offset>] (<reg>)

Display/Modify indirect memory which is pointed at by the "offset" to the register. The offset is in the range 0 to $FFFF. The register parentheses are required. Use the same register notation as the RM command. To exit, use ".<cr>".

MD <start adrs> [,<count>] or MD <start adrs> [<end adrs>]

Display "count" bytes of memory beginning at "start adrs". The right portion of the screen will contain ASCII equivalents of each byte or a period "." if the byte is non-ASCII. On hardware registers which accept only byte access (such as the 68901), a single line will be displayed.

MM <start adrs>

Display/Modify memory beginning at "start adrs". Enter ".<cr>" to exit. Avoid modifying DPR and local system variables.

BF <start adrs> <end adrs> <data>

Fill memory with "data" from "start adrs" to "end adrs".

BR <break adrs> [,<count>]

Set a break point address at a valid instruction with optional count. The default count is one and the maximum count is $FF. The count is intended for looping instructions. When the program is requested to G(o), it will execute through the break address "count" times then it will exit to the Monitor command level. Registers and memory state may then be examined or changed. If the PC and stack are not modified, another G(o) without an address will resume execution from the next instruction after the break point.

If preferred, the Trace mode will allow single instruction execution instead of the Go command with break points.

BR

Display the current break point address and optional count.

NOBR

Delete all break points.

T [<start adrs>] [,<count>]

Trace "count" instructions (default = 1) beginning at the current program counter or "start adrs". Trace will produce a register display similar to the RD command. If the count is for multiple instructions, XOFF/XON may be used to pause the display. The EXEC must be ON to allow the T command. (Send EXEC ON then ctrl-C to drop back into the Monitor).

G [<start adrs>]

Start execution from the current program counter or from "start adrs" up to but not including an optional break address. The EXEC must be ON to allow the G command. (Send EXEC ON then ctrl-C to drop back into the Monitor). While running local code, ctrl-C will exit to the Monitor if the local interrupt and system variables are not altered. See local code information.

EXEC ON

Go to the EXECUTIVE level and transition to the Awake Mode. See description. A 'EXC>' prompt will display. No further commands will be accepted at the serial port except the ctrl-C to return to the Monitor. CR will give another 'EXC>' prompt.

EXEC OFF

Go to MONITOR level and to the Sleep Mode. If the Executive is off, the G command with no address will respond with '?' and will not allow execution. DPR commands will be ignored.

Control-C (ETX or $03)

If at the EXECUTIVE level, drop down to the MONITOR level but stay Awake. The EXECUTIVE active state will be saved. To resume the Executive, do not change the PC or the stack. Then use the G command with no address. See Executive description. While running code from the Monitor level, ctrl-C will exit back to the Monitor if the local interrupt table, 68901 interrupt registers and system variables are not altered.

AD <start addr> [<final addr>]

Display continuous A/D conversions. Control-S (XOFF) or space will pause the display and control-Q (XON) will resume. <cr> will exit command. Addresses are in hex.

SRDL

Download S2 records through the auxiliary serial port at 4800 baud, 8 data, no parity, 1 stop bit. The ASCII S2 records are converted to an executable binary memory image and loaded at the designated address in the S2 records. The S2 records must contain exactly one control character (either CR or LF) between records. By varying the load address, subsequent downloads will overwrite previous RAM image. The first S7, S8 or S9 record encountered will exit to the Monitor. The loaded program will not automatically begin execution. Control-X will abort if the download locks up. Checksum errors are flagged and will abort the download. Avoid terminal keyboard entries during the download to prevent interrupts to the software UART timing loop. See description.

CT <adrs>

This is a hardware diagnostic command which continuously writes an incrementing code at any byte location in local memory. The location must be writable on byte addressing and can be either memory or hardware registers. A byte will be rapidly written sequencing from $00 to $FF, $00, $01, etc. An oscilloscope or logic analyzer may be used to trace address and data paths.

TC

This hardware diagnostic command continuously writes identical incrementing bytes to the start and final channel address registers. Byte codes cycle around from $00 to $FF, $00, $01, ,..., etc. While the addresses are unequal, the End of Scan (EOS) comparator will be false. The EOS will be true while both bytes are identical. Thus the EOS bit will rapidly toggle. The channel addresses will be distributed out the J3 analog channel expansion bus for diagnosis on slave MUX boards.

Control-X (CAN or $18)

Hard abort. This command jumps to initialization code and resets the DVME-601 firmware. Memory is retested and previous memory image may be overwritten. Control-X responds with a new prompt after selftest. Use ctrl-X to recover from a serial download which does not complete or to regain control from other locked up code. Use ctrl-C for a soft reset which preserves most registers and the Executive. The local interrupt table, 68901 interrupt registers, stack and local system variables must be intact for control-X to work. If ctrl-X fails, a hardware bus reset will be necessary.

ESCape (control-[or $1B)

Clear the command buffer and issue a '$' prompt. The Monitor will accept a new command. The previous command will not be erased on the display but will have no meaning.

3.6.4 Display/Modify Commands

The Display/Modify commands (MM, RI and RM) display a memory location or register and its current contents. Avoid modifying the system variables at the top of DPR and the internal system variables until you understand them. Three possible actions are available once a modify command is started:

1. Entering a hex data value terminated with (cr) will load that value into the last memory location or CPU register that was displayed. The next location or register will then be displayed. For the CPU registers, the display will cycle around through all the registers using either the D0-A7 set or the PC set.

2. Entering (cr) alone without a hex data value will display the next register or location without modifying the contents of the last one.

3. Entering a period and <cr> ".<cr>" will terminate the Display/Modify command without making any modification to the last value. A new Monitor prompt will be displayed.

3.6.5 Command Familiarization

The Monitor will be used extensively when developing your own programs. Before leaving this section, try a few commands. The H<cr> command gives you the Help menu. The RD<cr> command shows you the local CPU registers. The command MD 40000<cr> displays 256 bytes of the DPR memory starting from its BASE address. All addresses and data must be in hexadecimal. Remember that all addresses are relative to the local DVME-601 CPU. Any ASCII characters you see on the right portion of the display were probably left over from power up testing and have no meaning.

Connect a dc signal source such as DATEL's DVC-8500 Calibrator to channel 0 of the analog input connector, J1. Now give the A/D conversion command, AD 0<cr>. The display will rapidly scroll continuous A/D conversions as four ASCII hex digits. Recall that the data will be left justified so that the least significant digit will be zero for 12-bit A/D converters. Display scrolling can be stopped and restarted by pressing the space bar. The A/D conversions are being sent to the serial port rather than out the DPR. The Monitor program is locally triggering repeated A/D samples.

Try varying the calibrator to see changes in the displayed A/D data. This is the method used to adjust the zero and full scale adjustment potentiometers on the A/D converter. You may repeat this test with more than one channel connected. The command, AD 0 7<cr> will display eight channels separated by a blank line. The display will be more stable if only one channel is selected at a time. Channels with no input connection will display meaningless data. A carriage return will stop conversions and exit to the Monitor.

These tests demonstrate the DVME-601 operating as a stand-alone microcomputer making A/D conversions. The task ahead is to transfer blocks of raw or preprocessed A/D data at high speed out through the DPR, controlled by your host program.

3.6.6 Hardware Reset

During your evaluation of the board and especially when debugging downloaded programs, errant code may occasionally lock up the local CPU. Normally control-X will recover. If the interrupt system in the 68901 is altered, the DVME-601 may not respond to the terminal and further operation will be halted. The Executive software reset commands from the DPR may also recover control. Both the DPR commands and the serial port control-X force a hardware interrupt in the 68901 controller which autovectors the DVME-601's 68010 CPU to reset code in the EPROM - see Section 4.2.2.

If you have not yet written control code which is commanded through the DPR, DVME-601 program lock-ups may be recovered with a hardware bus reset. The DVME-601 responds to the bus SYSRESET* signal on P1-C12 by restarting its power-up Monitor sequence. On Motorola hosts, the SYSRESET* signal is initiated by a reset pushbutton on the bus controller. The bus reset will initialize both the host CPU and peripherals including disk controllers. Therefore the operating system will have to be rebooted. On some non-Motorola hosts, there is only one master reset switch.

The powerful 68000-series exception processing capabilities allow the host CPU to reset the bus under program control (the RESET instruction) when in the supervisor state. Your application code would do this if it detected loss of DPR control of the DVME-601. This would be an extraordinary condition but it is possible under harsh, high-noise operating environments.

PROGRAMMING INTRODUCTION

4.0 General Procedure

The DVME-601 operates as an intelligent slave to the host. After initialization, your supervisory host program loads a list of DVME-601 Executive commands and subroutine addresses in the DPR. The host starts command execution by writing to a reserved DPR hardware register which forces a local command interrupt. The 601 informs the host of command progress using DPR status words. The host then retrieves A/D data blocks through the DPR for further host processing.

Some Executive commands execute and stop quickly whereas others run indefinitely until a terminating condition occurs. Data ready events can be communicated to the host either via VMEbus interrupt or DPR status bits. The DVME-601 is a VMEbus slave and requires an external VMEbus CPU board to transfer commands and data blocks. The DVME-601 is not a VMEbus master and cannot control the bus.

From the dual port RAM, the 601 is either asleep (EXEC OFF) or awake (EXEC ON). If awake, the 601 is either in a ready command state or is running a list of subroutines. However while running subroutines or commands, the 601 may be stopped at any time and returned to the ready command state.

Simultaneous, concurrent operation and DPR duty cycle

During A/D scanning, other host tasks may take place at the same time. The host may read previous data from the DPR while the DVME-601 is writing data to another section of the DPR. The read/write access requests from either side of the DPR are arbitrated by high speed addressing logic. This DPR address steering logic grants access to the first side requesting a transfer and sends a short DTACK* signal to make the other side wait if there is a conflict. During the short DTACK*, the waiting CPU (either the 601 or the VMEbus CPU) executes NOP's. The short DTACK* is cycled with each transfer instruction.

During possible simultaneous access, moderate duty cycle should be used when accessing the DPR from the host so that the DVME-601 local CPU is not continuously locked out. Since the local 601 CPU controls the A/D and transfers A/D samples, DPR lockout delays might cause loss of local A/D samples. This moderate duty cycle includes host polling of DPR status bits. The DPR should be free for a few microseconds after each VMEbus access. A few NOP's or short counter loops may be inserted in host assembly language polling routines to provide this duty cycle.

The local 601 CPU requires 4 wait states at 8 MHz (500 nanoseconds) for each DPR word access plus the instruction time. Therefore the VMEbus DTACK* is periodically released for host access. Whenever there is no DPR contention, the VMEbus may use the DPR at full bus bandwidth. The 601 toggles the VMEbus DTACK* about every 200 nanoseconds in case a host DMA controller is used to block transfer the DPR. The VMEbus may of course access other non-601 portions of memory at full bandwidth while the 601 is accessing the DPR.

A/D Data Block Steering

A/D scans may be sent first either to a local RAM buffer or to the DPR. If a local math program will be used first, the raw A/D data should be sent to local RAM first. Block transfer subroutines are available between any two sections of memory (including local EPROM, RAM and DPR). They may be used to avoid the requirement for moderate duty cycle DPR access by interleaving block transfers using status bits.

Software Sequence

The overall software procedure to operate the DVME-601 is outlined below and will be detailed in the following sections:

1. After host power up, if you plan to have the DVME-601 send VMEbus interrupts, your host program should initialize the DVME- 601 interrupt ID vector register. This register appears as a memory address in the DPR and may be loaded by writing from the host to location BASE + $0FFFA. You may leave the interrupt disabled temporarily until your host is ready to accept interrupts from the DVME-601. The interrupt powers up on the board in the disabled mode and may be controlled by your host program through bit 7 of DPR word location Base + $0FFF8.

2. At power up, the DVME-601 performs self test, lights its LED lamp and waits for a WAKEUP command. The board will be in its SLEEP state with its Executive firmware OFF. It will ignore all DPR activity occuring from the host side except a WAKEUP command. Your host program should transition the DVME-601 to its AWAKE state. This is done by writing the Executive ON command into the DPR and interrupting the DVME-601 to execute the command. The local interrupt occurs with a host write to DPR location BASE + $0FFFC on word access. The board will acknowledge the Executive ON state with status words in the DPR which your host program should confirm.

3. Your program may now load A/D subroutine addresses and commands and interrupt the board to begin execution. As the board transfers A/D data blocks to the DPR, it will flag the host with a data ready DPR status bit and maskable VMEbus interrupt. Your host program block transfers the A/D data to other system memory or operates directly on the DPR data. If you have also loaded the DVME-601 with a preprocessing program which operates on the raw binary A/D data, that program would execute after A/D scan collection and before transfer to the host.

4. After each A/D block transfer, your host program should further process the data. This may include math, mass storage, communication, display or control. The DVME-601 may begin the next scan immediately while the host does its processing.

5. After all A/D data is taken, the host terminates the session. If required, the DVME-601 Executive may be turned OFF to ignore the DPR.

4.1 Operation with No Local User Program

The DVME-601 will function without having to write, load and debug any local user programs if no local math is needed. Operation will run entirely from EPROM using the Executive firmware supplied with the board. Your host program will fully control the Executive including how it transfers A/D data to the DPR. The Executive firmware is suitable for most applications but enhanced performance is possible if you execute your own local program in the DVME-601.

To help you decide if your own local programming is appropriate, examine the total system throughput which your application requires. Compare this to the analog input bandwidth expected and the DVME-601 throughput rates. Determine the required periods for math or other processing on the A/D data. You will need timing information on other system I/O peripherals such as disk or display update rates managed by the operating system. Fast math should be done on a host CPU with a math coprocessor.

Simultaneously scanning DVME-601 analog channels while the host performs other tasks significantly offloads the host. Doing A/D math preprocessing with your own program loaded into the board may offer additional host offloading and higher system throughput.

You should avoid local programming if you do not need it, or do not have the time or resources to develop the program. You should be proficient in 68010 assembly language to write the program even if you choose to write most of the code in a high level language.

4.2 Memory Map and Internal Architecture

Although the DVME-601 internal hardware registers and CPU resources are fully managed by the Executive firmware, an understanding of these resources will clarify operation of the Executive. The host program which you will develop must use this memory map for commands and data.

Figure 4.0 shows the Memory Map for the local 68010 microcomputer and the DPR. Addresses on the left side of the figure are relative to the local 68010 CPU. Addresses on the right side are relative to your host. The DPR area has two sets of addresses, corresonding to the local CPU and to the host. The BASE address is that address which you selected with the on-board jumpers back in Section 2. The DPR area should be carefully compared to the typical host map in Figure 2.0. Figures 2.0 and 4.0 overlay each other in the DPR area.

Certain areas of this memory map will receive greater attention later in this section. These include the local hardware registers, the DPR data transfer area and the DPR command/status area mapped adjacent to the interrupt hardware registers.

Local Memory		Host Memory
$000000 to $00FFFF	64 Kb Local EPROM or lower 64 Kb of 128 Kb EPROM (Read only)	
$010000- $01FFFF	Upper 64 Kb of local EPROM if 128 Kb installed	
$020000- $02FFFF	64 Kb Local RAM (R/W)	
$040000- $04FFF7	64 Kb Shared Dual Port RAM (R/W)	BASE + $0000-BASE + $FFF7 Shared 64 Kb DPR (R/W) [$FFF0-FFF7 are soft-mapped Read/Write command/status]
$04FFF8- $04FFF9	Not used	BASE + $FFF8-BASE + $FFF9 Enbl.intrpt.to VMEbus (R/W)[bit 7]
$04FFFA- $04FFFB	Not used	BASE + $FFFA-BASE + $FFFB Host intrpt. ID Vector (R/W)
$04FFFC- $04FFFD	Not used	BASE + $FFFC-BASE + $FFFD Force command intrpt to 601 (Wr-only word from host)
$04FFFE- $04FFFF	Not used	BASE + $FFFE-BASE + $FFFF Not used
$06XXXX	A/D Start Chan. Addr. Reg. (Wr)	
$08XXXX	A/D Final Channel Address Reg. (Wr)	
$0AXXXX	Cmd/Stat Reg.(R/W) [EOC,EOS,LED,A/D,etc.]	
$0CXXXX	Start A/D Convert (Write only)	
$0EXXXX	Force intrpt.to VMEbus Host (Wr)	
$10XXXX	A/D Data Register (Read only)	
$120000- $12002F	68901 Intrpt/Timers/ USART/Para.Port (R/W)	

Notes:

1. Unlisted addresses are redundant or not defined. All addresses are in hexadecimal. "XXXX" bytes are not decoded and are don't care. Factory-jumpered BASE address is $FA0000.
2. Hardware registers from $6XXXX to $10XXXX require MOVE.W instructions. Local RAM from $20000-20FFF is reserved. 68901 registers require MOVE.B instructions or Read-Modify-Write.
3. BASE + $FFFC is write-only from VMEbus and should be located beyond power-up memory testing.

Figure 4.0 DVME-601 Memory Map

4.2.1 Dual Port RAM Organization

Figure 4.1 shows the DPR in more detail from the host side. The dual port RAM consists of normal read/write random access memory and memory-mapped hardware registers. These registers support interrupts to both the DVME-601 and to VMEbus. The registers should be accessed from the host with word (.W) instructions only.

The DPR read/write memory section is further subdivided into a small command/status area and a large data transfer area. Function commands, subroutine addresses, A/D data block uploads, and optional S record program downloads occupy the data transfer area from BASE + $00000 to $0FFEF relative to the host. Executive commands manage this area but it can be redefined by local user programs. Transient longword (.L) function command blocks build downward from longword address BASE + $0FFEC and are normally less than 100 bytes. They may be overlaid by A/D data once the command starts execution. Typically, A/D data blocks build upwards from BASE + $00000. The Executive uses one or two buffer pointers to manage the destination addresses of A/D data blocks. Figure 4.2 shows both A/D data and function command blocks. They will explored in greater detail after the A/D command modes are discussed.

The DPR area from BASE + $0FFF0 to $0FFF3 is presently not assigned and may be used by your local DVME-601 program. The command/status area from BASE + $0FFF4 to $0FFF7 contains software control variables and system information. This area is defined by Executive firmware and is only active while the board is in the Executive state. Normally, the local CPU does not execute code from the DPR.

BASE + $00000 to BASE + $0FFEF (Read/Write)	Data transfer area (A/D data blocks, S record downloads, function blocks and commands, subroutine addresses and parameters). [Commands build downward from $0FFEC on longword access].
BASE + $0FFF0 to BASE + $0FFF3	Reserved command/status area (Not used by Executive. Available to user programs). Read/Write.
BASE + $0FFF4 BASE + $0FFF5	A/D Status word (used by Executive). ADSTAT, firmware defined. Read/Write.
BASE + $0FFF6 BASE + $0FFF7	DVME-601/Host Status word (used by Executive). HSTSTA, firmware defined.
BASE + $0FFF8 to BASE + $0FFF9 (Read/Write)	Enable interrupt to VMEbus on word (.W) access. If bit 7=1, interrupts are enabled. If bit 7=0, interrupts are disabled. Bit 0 is RESET.
BASE + $0FFFA BASE + $0FFFB	VMEbus interrupt ID register (bits 0 thru 7 on word access, Read/Write). Bits 8-15 are don't care.
BASE + $0FFFC BASE + $0FFFD	Force local interrupt to DVME-601. Write-only on word access. All bits are don't care. Executes commands.
BASE + $0FFFE,F	Not used

Figure 4.1 VMEbus Dual Port RAM Memory Mapping

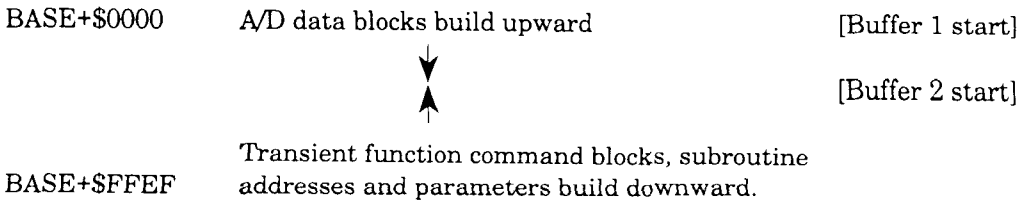


Figure 4.2 DPR Data Transfer Area

4.2.2 Interrupt Registers

Interrupt hardware registers are memory-mapped in the DPR and in local memory. They are shown in Figure 4.3 and 4.4 from the host side. The host must write an interrupt ID vector at host DPR location BASE + $0FFFA before VMEbus interrupts may be enabled. Interrupts to the VMEbus are disabled if the host resets the interrupt enable register at DPR BASE + $0FFF8.

The host writes to location DPR BASE + $0FFFC to force a local interrupt to the DVME-601. The register write generates an autovector from the 68901 interrupt controller. This causes the Executive to load new function command blocks or commence execution of previously-loaded commands.

Local DVME-601 programs (including most Executive A/D commands) may interrupt the VMEbus by writing to local address $0EXXXX (Figure 4.5) if allowed by the VMEbus interrupt enable register. All registers must use word (.W) access.

SOFTWARE RESET

Beginning with artwork revision G, DVME-601 boards will accept a control bit to force a hardware reset of the board and local CPU from the DPR. A "1" bit written to bit 0 in the VMEbus Interrupt Enable Register (BASE+$FFF8,9 in word mode) will force a reset. PAL U3 and the gates 601.abl file were changed for this function.

Compatibility: This change is totally downward compatible to all existing DVME-601 boards which do not use bit 0. No EPROM change is made.

Purpose: The hardware reset will allow a user's VME host program to reacquire control of the DVME-601 if noise or other factors have caused a lockup of the 601's local 68010 CPU. The reset bit can be invoked without causing a reboot and reset of the entire VMEbus host system. A lockup would be indicated if the 601 failed to respond to command interrupts from the host.

The 601's VMEbus reset line (SYSRESET*, pin P1-C12) is still available and will also cause a hardware reset of the 601. Normally, SYSRESET* also causes a reboot of the entire VMEbus host.

After a hardware reset toggle, the 601 begins self-test and enters the Monitor, verified on the J2 serial port terminal. The Exec will be off. The host may reacquire control through the DPR by turning on the Exec and running AFB's.

Usage: The reset bit is a latching function, therefore the controlling host program must write a "1" to cause the reset. Then a "0" must be written to allow the 68010 CPU to run. Allow at least 10 microseconds between these two writes for the reset state to propagate through the 68010. Remember to correctly encode the VMEbus Interrupt Enable (Bit 7) whenever writing to BASE+$FFF8,9.

Read/Write BASE+$FFF8,9 VMEbus Interrupt Enable Register (Defaults to $0000 at power-up)

15 – 8	7	6 – 1	0
not used	Enbl Int.	not used	RST

0 = Disable Interrupts
1 = Enable Interrupts

0 = Run
1 = Reset CPU

Figure 4.3 VMEbus Interrupt Enable Register

Interrupt ID Vector Register (Read/Write)
[Address = BASE + $0FFFA]

15	14	13	12	11	10	09	08
X	X	X	X	X	X	X	X

Interrupt ID Vector Register

07	06	05	04	03	02	01	00
Vector Bit 07	Vector Bit 06	Vector Bit 05	Vector Bit 04	Vector Bit 03	Vector Bit 02	Vector Bit 01	Vector Bit 00

Register to Interrupt Local CPU (Write only)
[Address = BASE + $0FFFC]

15	14	13	12	11	10	09	08
X	X	X	X	X	X	X	X

Register to Interrupt Local CPU

07	06	05	04	03	02	01	00
X	X	X	X	X	X	X	X

"X" bits may be 0 or 1 (don't care).

Figure 4.4 DPR Interrupt Registers from VMEbus Side

VMEbus Interrupt Register (Write only)
[Address = $0E0000 - $0FFFFF]

15	14	13	12	11	10	09	08
X	X	X	X	X	X	X	X

VMEbus Interrupt Register

07	06	05	04	03	02	01	00
X	X	X	X	X	X	X	X

"X" bits are don't care.

Figure 4.5 Local Register to Interrupt VMEbus

4.2.3 Local Registers

All local DVME-601 hardware registers are memory mapped and are managed by the Executive. They consist of data acquisition registers, the VMEbus interrupt register and peripheral I/O registers in the 68901 controller. If you use the Executive, register details are not important except for the A/D command register.

Addressing for all local registers except the 68901 registers does not decode the least significant 16 bits and are shown as "X" (don't care) in Figure 4.0. Word access (.W) must be used with the A/D and VMEbus interrupt registers and byte access (.B) for the 68901 peripheral registers.

The 68901 controller is extensively used by the Monitor and Executive firmware for the serial port, baud rate clock, A/D pacer timers and local interrupts.

4.3 Data Acquisition Registers and General Procedure

The A/D section registers include write-only start and final analog channel address registers, an A/D converter start register, the A/D data register and an A/D command mode/status register. Whether you use the Executive or your own local program, the general procedure for operating the A/D section is as follows:

1. Setup

After deciding how the DVME-601 will be integrated into your host as discussed in Section 3, configure and install the board. Make all analog input, channel expansion, Monitor terminal and trigger/timer connections. Check out the board using your RS-232-C terminal and the firmware Monitor, also discussed in Section 3. Decide how you will handle interrupts but leave them disabled until you have an interrupt handler program available in your host.

2. How Many Samples?

Decide how many samples you will take before data transfer. Determine what will stop the sampling process. The choices are one sample, one scan of multiple sequential channels, N scans or scan continuously. The host can stop the sampling process at any time. The DVME-601 becomes more efficient with larger numbers of samples. But large scan blocks may force the DVME-601 to be idle waiting for the host. And long multiple scans may be inappropriate for certain applications such as graphics display updates or disk buffers that are too long. Scan termination is handled by the Executive commands. Filling either the local RAM or the entire DPR will take about 30,000 2-byte samples maximum.

Continuous scanning which does not miss an A/D trigger requires swapped dual DPR buffers and a host which can keep up with the 601. The buffers are switched using software address pointers. Buffer ready signals are most efficient if the 601 interrupts the VMEbus with each buffer switch. Normally the host will then build a very large contiguous host data array or store sequential samples on hard disk. Refer to the DVME-601 application floppy disk for an example program using high speed buffer switches.

3. Optional User Preprocessing Program

If you plan to run a local preprocessing program, decide when it will execute, depending on step 2 above. The choices are to process after each sample, after a scan of samples or after N scans. Keep in mind that memory is required to hold output (processed) data unless you wish to overwrite raw input data. The preprocessing takes place before transfer to the host through the DPR. You may need to use the DPR for temporary data storage.

Cross channel computation (comparing one sample against others) requires that you sample a full scan or multiple scans before processing. You should develop your application without any preprocessing first. Then write the program, load it into the DVME-601 and debug it. Refer to Section 5.

4. Channel Addressing

Decide how to address the analog channels. The choices are single channel, sequential (scan) addressing, or random addressing from a table. Random addressing is best done with your own local program but it is possible using only the Executive. Random addressing may be used to sample higher bandwidth channels more frequently than slow channels, if the input frequencies are well characterized.

For multiple sequential scans (N scans), automatic reloads of the address register from the original start channel occur when the final channel is reached. This avoids a delay having the local CPU reload the start address. This feature is included in multichannel scan Executive subroutines.

Once you have determined channel addressing, load the start and final channel address registers. Then program the A/D command register for the proper mode.

5. A/D Start and Data Transfer Method

Decide the A/D conversion start method and how data transfers will occur to local memory. Data transfers occur either by a polled EOC status or the fast throughput method discussed below.

The start methods include a register write, by reading the A/D data register or by an external TTL trigger. The trigger can come from an external event or from a DVME-601 timer output from the 68901 controller. An external jumper connection from the timer is made on the J2-10 connector.

Executive commands will let a trigger start one A/D sample, or one scan. One or more timers can be cascaded for very long data logging applications. If you use the timer, program the timebase frequency with the appropriate Executive command. The Executive loads the timebase into a local RAM variable then loads the 68901 timer registers when an A/D sampling subroutine starts. The Executive stops the timer after the sampling is done. For your own local A/D code, bit 2 in the A/D command register may be used to stop and start the A/D. Or you may follow the EPROM subroutine examples.

If the application is for a high bandwidth single channel, you should consider the fast throughput data transfer method. Otherwise the status method is more appropriate. Both are programmed through the A/D command register via Executive commands.

6. A/D Data Memory Destination

Decide the memory destination of A/D samples. The choices are to send all data to the DPR, to send it to local RAM first, or to use two swapped DPR buffers. This programming is best done with the Executive. Samples should be sent to local RAM first if the data will then be preprocessed before transfer to the DPR and the host.

The Executive includes functions to assign buffer addresses, manage sample transfers to the buffers and terminate the sampling when buffers are full. These functions also post status bits and optional interrupts. They manage the dual buffer mode with automatic swaps of addresses and status bits to identify the active buffer.

7. Host Data Transfer Synchronization

Decide how the host will be synchronized for scan transfers. The choices are status bit, status-plus-interrupt or interrupt-only. The status bits are set by the DVME-601 in the DPR status area. They are set when an A/D data block is ready for transfer out of the DPR to the host. When the host transfers the data block successfully, it should reset the status bit. Executive firmware offers the option of having the DVME-601 either poll this bit or ignore it. If the bit is polled, the DVME-601 will wait for the host to reset the bit before starting more data sampling. If the DVME-601 waits, the host will not lose any data. But input bandwidth may suffer if the host does not respond quickly.

If the DVME-601 does not wait for the host to reset the status bit and verify data transfer out of the DPR, continuous sampling is possible using either a single buffer or two software-swapped buffers. If it is done using all Executive commands and subroutines, a brief Executive software overhead interval of a few hundred microseconds will be encountered between multi-sample A/D subroutines.

If the buffer switching is done using the fast buffer switch method described below, high speed non-stop sampling will be possible with virtually no delay between buffer switches. VMEbus interrupts signal the buffer switches. Both applications remain under Executive control.

8. Your Host Program

You must write a program in the host to operate the DVME-601. The program will send Executive commands, start the scanning, retrieve A/D data blocks and pass them on to other application programs.

The decisions listed above will now be explored in the following sections as the registers are programmed.

4.3.1 Analog Channel Addressing

Address sequencing logic uses the start and final analog channel address registers for multichannel scans and automatic End of Scan detection. The channel address registers are shown in Figure 4.7. The current channel address is the address code that is actually applied to the analog input multiplexer. The current channel address register is initialized by writing to the start channel address register at location $06XXXX. This address will automatically increment at the End of Conversion if the autoincrement bit (A/D command/status register bit 3) is set to 1. The autosequencing of the channel address relieves the local CPU of having to control channel addressing and increases throughput. A final channel address register is located at memory address $08XXXX.

If the channel address is not allowed to increment (A/D command bit 3 = 0), the system will convert only a single channel. The channel address selected will be the last current address. Fast settling will result since the address code to the multiplexer does not change.

Multi-sample data blocks are first started by a write to $C0000 or an external trigger. Subsequent samples are started by reading the A/D data register. In autoincrement mode, when the current channel increments so that it equals the address stored in the final channel register, the End of Scan bit 8 is set in the A/D status register at the rising edge of EOC.

Channel addresses 0 through 15 are used for on-board single-ended inputs on the DVME-601. Channels 0 through 7 are for on-board differential inputs. Addresses 16 through 255 are for expansion inputs controlled through the J3 channel expansion bus. This bus connects to DATEL DVME-64X series slave multiplexer boards adjacent to the DVME-601. Please refer to Figure 4.6.

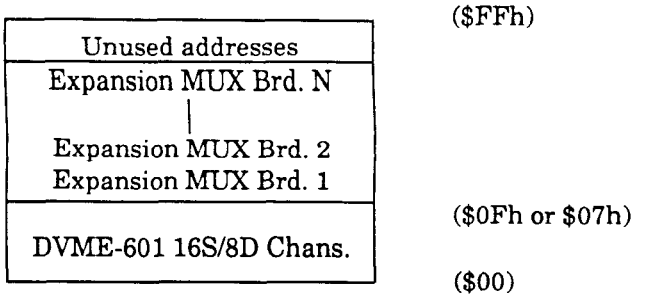


Figure 4.6 Analog Channel Expansion Address Map

A/D Start Channel Address Register (Write only)
[Address = $060000 - $07FFFF]

15	14	13	12	11	10	09	08
X	X	X	X	X	X	X	X

A/D Start Channel Address Register

07	06	05	04	03	02	01	00
STCH 7	STCH 6	STCH 5	STCH 4	STCH 3	STCH 2	STCH 1	STCH 0

A/D Final Channel Address Register (Write only)
[Address = $080000 - $09FFFF]

15	14	13	12	11	10	09	08
X	X	X	X	X	X	X	X

A/D Final Channel Address Register

07	06	05	04	03	02	01	00
FNCH 7	FNCH 6	FNCH 5	FNCH 4	FNCH 3	FNCH 2	FNCH 1	FNCH 0

"X" bits are don't care.

Figure 4.7 Local A/D Channel Address Registers

A/D Data Register (Read only)
[Address = $0100000 - $011FFFF]

15	14	13	12	11	10	09	08
BIT 1 (MSB)	BIT 2	BIT 3	BIT 4	BIT 5	BIT 6	BIT 7	BIT 8

A/D Data Register

07	06	05	04	03	02	01	00
BIT 9	BIT 10	BIT 11	BIT 12	BIT 13	BIT 14	BIT 15	BIT 16 (LSB)

Figure 4.8 Local A/D Data Register

4.3.2 Left Justified A/D Data

Figure 4.8 shows the format for the local A/D data register at location $010XXXX. Executive firmware transfers binary A/D samples from this location to either local RAM or the DPR, depending on the buffer mode. A/D data for all DVME-601 A/D converter options is left justified with the most significant A/D data bit at bit address 15. This allows equivalent scaling and sign detection between different converters. Users may write one set of programs and change converters without major program rewrites. The unused bits below the least significant bits for 12 and 14 bit converters are zeroes.

For 12-bit A/D converters (models DVME-601A, E and B), bit 4 is the least significant bit. For the 16-bit A/D converters (model DVME-601C and D), bit 0 is the least significant bit.

Note - A/D converter notation labels the Most Significant Bit as bit 1, unlike most computer registers where the MSB is bit 15. This is based on the origins of successive approximation A/D conversion where the first decision bit selected in the SA register corresponded to the largest analog input weighting. For binary A/D converters, bit 1 is about half of full scale.

4.3.3 A/D Command/Status Mode Register

Location $0AXXXX in the DVME-601's local memory map contains a read/write portion and a write-only A/D command register (bits 5 through 0). Please refer to Figure 4.9. The register is managed by Executive firmware and is presented here so you will understand A/D modes in detail even if you do not write any local programs. If you decide to write local programs, you may either directly control this register or use the Executive subroutines.

All memory operations to location $0AXXXX should be on word (.W) access. On write, bit 15 turns the front panel LED lamp on or off. The Monitor firmware turns this lamp on after successful power-up self testing.

4.3.3.1 EOS and EOC Status Bits

On read, command register bits 15 and 8 contain the A/D converter's End of Conversion (EOC) and End of Scan (EOS) status bits respectively. EOC is cleared to 0 during conversion and is set to 1 when conversion is done. The EOC transition to 1 also indicates that a new channel address has been selected if the increment mode is enabled. Valid A/D data may be read in location $010XXXX when EOC = 1. EOC and EOS are cleared to 0 upon reading the A/D data register.

EOS is set to 1 whenever the current channel address register is equal to the final channel register and is clocked by the EOC.

The End of Scan (EOS) is a method to repeatedly scan a list of sequential adjacent channels while the address logic automatically sequences the addresses. The EOS signal is set when the current channel address register equals the final address register AND the A/D End of Conversion (EOC) occurs. At EOS, if the autoincrement mode is enabled (command 3 = 1), the current channel register (which controls the analog multiplexer) will be reloaded with the start channel, allowing a scan to start over. This starts the input circuits settling on the next analog value while the CPU reads the last A/D sample. High speed is offered with excellent analog specifications.

EOS is reset by the next EOC when the current address does not equal the final address. Normally this occurs when the scan starts over. EOS is also reset when the autoincrement command bit 3 is written with a zero.

Hardwired local interrupts are sent to the 68901 controller for both EOC and EOS. They are normally masked off by the 68901's masking registers. To retain high speed, the interrupt is not used by the Executive. The interrupt would be useful in a user's local preprocessing program doing data math while a low speed, high resolution A/D converter is making the next conversion.

4.3.4 A/D Conversion Start Methods

Write-only bits 5 through 0 of the A/D command/status register determine how the A/D converter will be triggered, how channel sequencing will occur and how data will transferred out of the converter. Please refer to Figure 4.10. A/D conversion starts with a short hardware settling delay inside the converter module then the actual conversion. The sequence may be started by three sources:

1. An external TTL trigger input. The logic enables the trigger and waits for the trigger falling edge. The trigger may be derived from an external event or the local Pacer clock output wired to J2-10.

2. When the local CPU reads the A/D data register ($10XXXX).

3. A local CPU write to the start conversion register, location $0CXXXX.

The start pulse may be used to start either a single conversion or a scan of N channels. If the start command mode defines a scan, then the Executive must still start individual samples by reading the A/D data register. The read data conversion start method must have been encoded for scans.

These modes may be partially combined as shown in Figure 4.10. The bottom portion of the figure explains the start mode bits 1 and 0 in more detail. Modes 01 and 11 are intended for multichannel sequential scans. In mode 01, the first conversion is started with a write to location $0CXXXX. Subsequent conversions occur by reading the A/D data register. Mode 11 is similar except that the first conversion is started by either the external trigger or the Pacer clock connected to the trigger.

For the external trigger modes 10 and 11, writing the command code arms the trigger by enabling the trigger gate. The converter then waits until the actual trigger. Note that mode 11 is designed to use one trigger to start a full scan. The local CPU polls EOC looking for a conversion. The trigger starts the first conversion and subsequent reads of the A/D data register convert the rest of the scan.

CAUTION - When using mode 11, externally disable subsequent triggers after the first one to avoid disrupting the data storage of remaining scan samples. Re-enable the trigger for the next scan or make sure triggers are held between scans.

Modes 00 and 10 should set command bit 3 to 0 to disable address increment. Modes 01 and 11 may be either single channel, random or sequential scan.

If at EOS, the rescan command bit 4 inhibits or enables conversion starts by data reads but has no effect on external triggers. Rescan = 0 for 1 trigger per scan. Rescan = 1 for 1 trigger per "N" scans or continuous single channel reads.

Local A/D Command/Status Register (Read/Write)
[Address = $0A0000 - $0BFFFF]

15	14	13	12	11	10	09	08
LED/ EOC Status	X	X	X	X	X	X	Read EOS Status

Bit 15 Write: 0 = LED off, 1 = LED on
Bit 15 Read: 0 = A/D not done, 1 = End of A/D conversion
Bit 8 Read: 0 = Current channel not equal to final channel
1 = Current channel equal to final channel

Local A/D Command/Status Register (CMD5 - CMD0 are write only)

07	06	05	04	03	02	01	00
X	X	Write CMD5	Write CMD4	Write CMD3	Write CMD2	Write CMD1	Write CMD0

(See command coding information for bits CMD5 through CMD0 in Section 4.3 and Figure 4.10).

Start A/D Conversion Register (Write only)
[Address = $0C0000 - $0DFFFF]

15	14	13	12	11	10	09	08
X	X	X	X	X	X	X	X

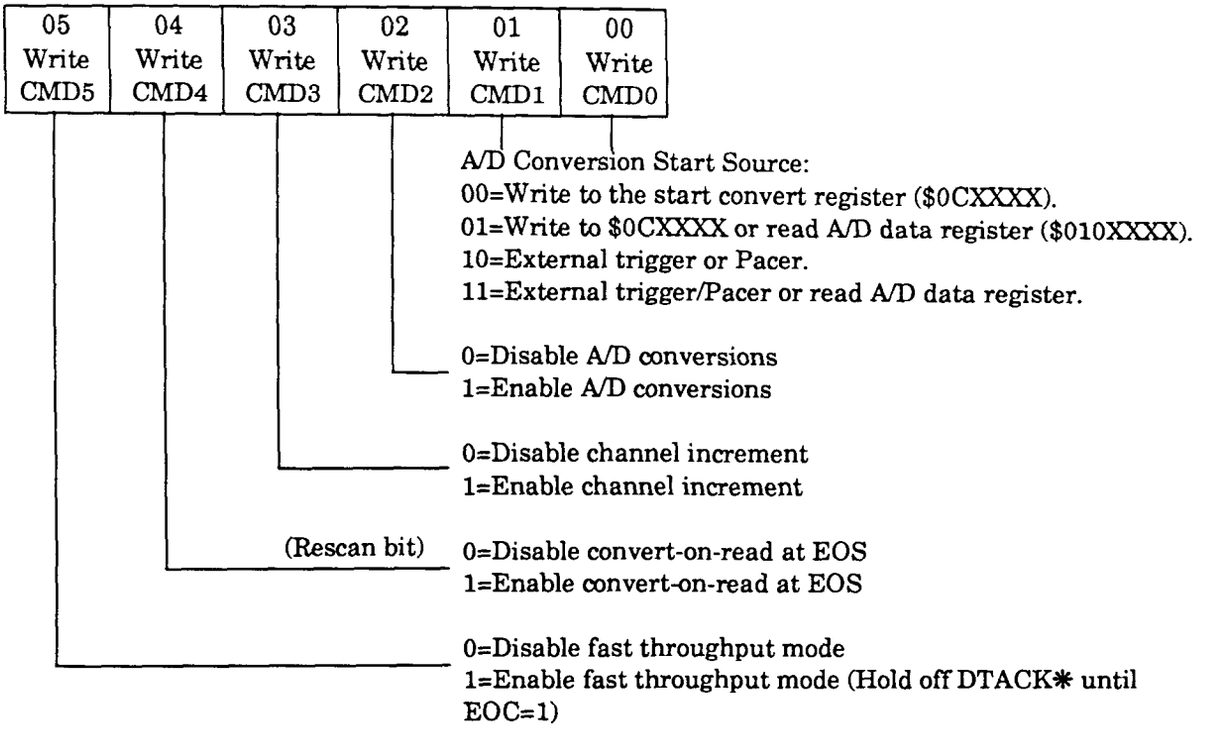
Start A/D Conversion Register

07	06	05	04	03	02	01	00
X	X	X	X	X	X	X	X

"X" bits are don't care.

Figure 4.9 Local A/D Command/Status and Start Conversion Registers

Local Command/Status Register [Address = $0AXXXX]



Command Bits 1,0	Start A/D Conversion Source			Recommended Channel Address Mode
	Write to $0CXXXX	External Trigger or Pacer	Read A/D Data Reg. $010XXXX	
00	X			Single channel or random.
01	X		X	Sequential scan.
10		X		Single channel or random.
11*		X	X	Sequential scan. 1 trigger/scan

*Note - Mode 11 is especially designed so that one trigger starts an entire scan.

Figure 4.10 Local A/D Command Register Decoding

4.3.5 A/D Data Transfer Methods

There are two methods of indicating when to transfer A/D data to local memory or to the DPR. In both methods, the Executive uses the A/D's End of Conversion (EOC) status bit. This bit is reset to 0 during A/D conversion and is set to 1 when conversion is done.

In the first data transfer method, the local CPU polls the EOC bit until EOC is done, then moves the data to memory. This EOC polling method is used for multichannel scanning and either local or external triggering.

The second method is called the fast throughput mode, set when A/D command bit 5 = 1. This mode delays the DTACK* signal to the local 68010 CPU when it attempts to read the A/D data register if conversion is not done. When the End of Conversion becomes true (EOC = 1), DTACK* is released. Then the local CPU immediately moves the data to memory or to the DPR. By combining fast throughput with the read data conversion start, the MOVE.W instruction simultaneously starts the next conversion. A very short instruction loop increments destination pointers and determines when to exit scanning. This offers high speed by using fewer instructions and avoiding EOC polling. See the program disk for a software example.

When used with the 68010 DBF loop instruction, the fast throughput mode acts like a DMA transfer. The instruction remains in the 68010's instruction queue and no opcode fetch cycle is required. The CPU is completely slaved to the A/D EOC timing. While DTACK* is held off, the local CPU is unavailable to other tasks. After the EOC goes high and during the brief settling delay before the next conversion, the CPU will respond to interrupts such as a host command request.

CAUTION: The fast throughput mode should only be used with a single channel input since there is little settling time allowed for channel switching transients. This mode would normally be combined with start mode 100001 which starts A/D conversions by reading the A/D data register at location $010XXXX.

External event or Pacer clock triggering (mode 100011) should be used with caution in fast throughput since the DVME-601 may ignore host command interrupts while waiting for a trigger. If the 601 locks up waiting for a trigger that never arrives, a host VMEbus RESET will be required. This usually reboots the system, so be careful! Combining fast throughput with convert-on-read is safer but cannot be used with external trigger synchronization.

The A/D start logic may be inhibited by setting command bit 2 to 0. This would be used to ignore external triggers until ready or for other usage.

Aggregate System Speeds

Total system speeds involve several simultaneous, partially overlapped events. A/D conversion may occur while the input stages are settling on the new sample, the previous sample is being transferred to memory and software destination pointers are being managed. Throughput depends on-

1. The trigger method (external trigger or convert-on-read-data).

2. The number of channels and PGA gain. (Fastest at gain=1)

3. The software data transfer method (polled or fast throughput).

4. The data destination (DPR or local RAM).

5. The total number of samples before block transfer.

The figures given in the specifications are raw A/D conversion periods from the falling edge of an external trigger (or the edge of a read A/D data command in convert-on-read mode) to the rising edge of the EOC. This period includes the switch from sample to hold mode, a short settling delay then the actual A/D conversion. The local CPU EOC polling and data transfer times must be added to this period. In fast throughput mode, these software times will partially overlap the next acquisition and settling delay. With efficient local coding, true single-channel sample-to-sample rates under 4 microseconds to memory may be achieved depending on modes. Refer to the example program disk.

Important: If channel increment mode is enabled, the channel address increments at the rising edge of EOC to start settling on the next channel. Multichannel inputs require at least 6 microseconds (gain = 1) after channel sequencing and before the next conversion start for proper settling.

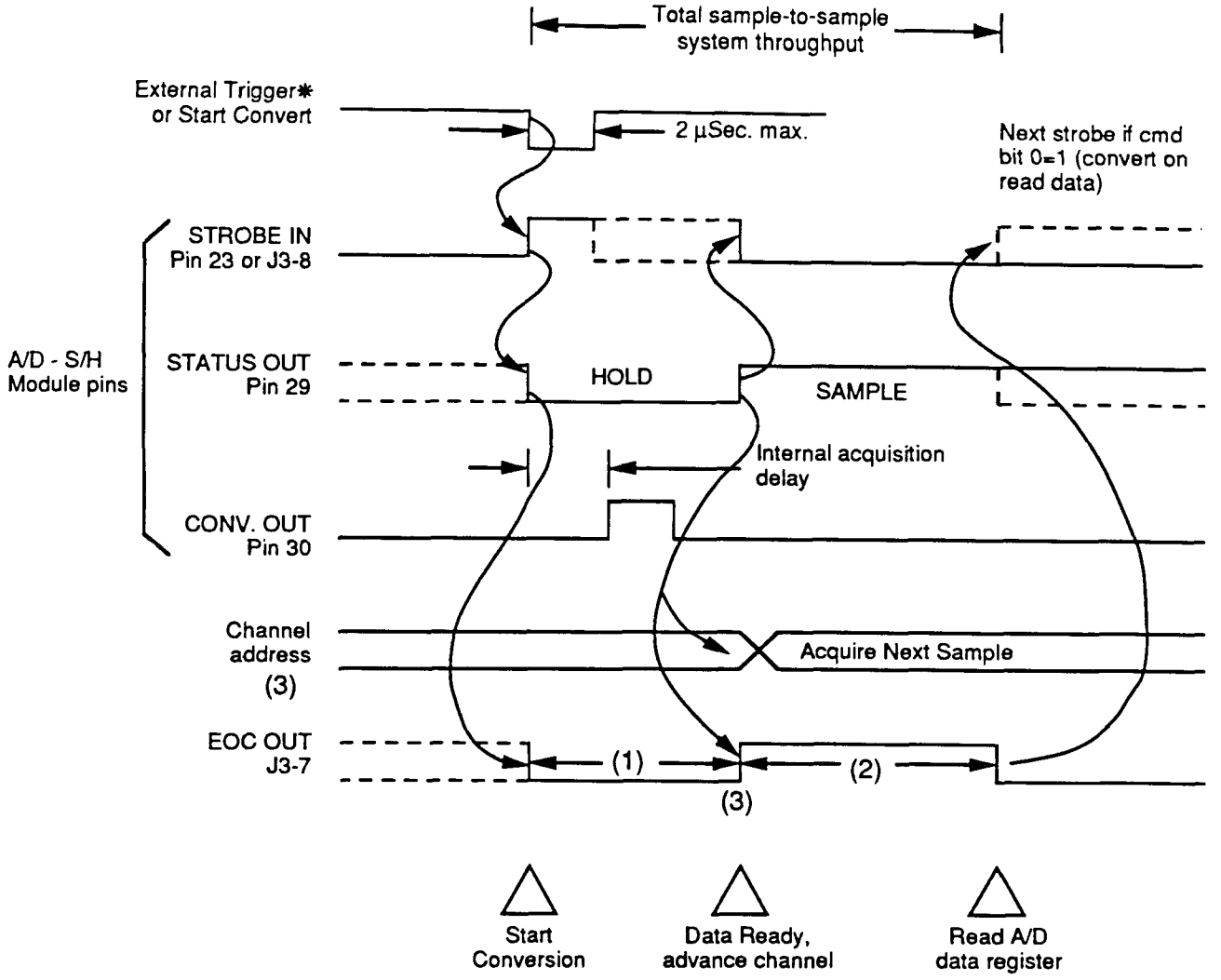
To obtain the highest speed, use the DVME-601E only in the single channel address mode (no channel address autoincrement) at a gain of one. Full scale sinusoidal input bandwidth of the DVME-601E (single channel, PGA bypassed) is 100 KHz typical, 40 KHz minimum at minus 65 dB harmonic distortion. Observe Nyquist trigger rate rules and input spectral content for DSP applications.

Filling the DPR or local RAM at over 250 KHz must pause at about 30,000 samples (60K bytes - memory full) then a brief block transfer to the host. Data may be steered to the DPR if there is low to moderate duty cycle contention for the DPR by the host.

If continuous, non-stop A/D triggering is required which runs "forever" with no sample loss, a special swapped overlapped buffer technique is available at approximately 110 KHz. See the program disk. Using the high speed DVME-601E 2 microsecond A/D, these times are summarized below:

Total Samples per Second	**Continuous Non-stop Sampling**	**Number of Channels**	**Software Method**
300 KHz (typ.)	<30,000 samples	one	Executive subroutines using fast-throughput until memory full. Brief block transfer to host then resume sampling.
110 KHz (typ.)	no limit (non-stop)	1 to 256	Special swapped buffer fast throughput (see the program disk)
170 KHz (typ.)	<30,000 samples	2 to 256	Executive subroutines (6 microsec. settling required between channels). Fill memory then brief block transfer to host. Then resume sampling.

The user must provide settling delays for higher PGA gains. Either the timer may be used or software delay loops.



(1) Published A/D conversion period.

(2) EOC polling software delay. This interval is essentially zero if command bit 5 = 1 (fast thruput), and the CPU attempts an A/D data read before EOC = 1. EOC resets to zero at start of conversion or a read of the A/D data register.

(3) Channel address advances at rising edge of EOC if command 3 = 1 (autoincrement). Allow 6 microseconds settling (gain = 1) until next conversion start in multichannel only.

Figure 4.11 A/D Timing Diagram

PROGRAMMING THE EXECUTIVE

5.0 Introduction

The Executive is part of the DVME-601's EPROM firmware. Using a DPR command interrupt, it manages up to two user-defined functions in real time. The Executive processes host commands and built-in subroutines, manages A/D conversion activity, transfers A/D data blocks to the DPR, communicates with the host and with the serial port. It also supports local user-written subroutines, either downloaded or in reprogrammed EPROM.

Two States (Awake or Asleep)

The Executive has two states, either ON (Awake) or OFF (Asleep). These states may be enabled from either the Monitor serial port or from the DPR. Optional user-written code may be run from either the serial port or the DPR (with some differences). However, Executive commands can only be loaded and run from the host through the DPR. Access from the Monitor serial port is not required to run the Executive.

At power up, the Executive is OFF and must be turned ON via software before any other commands. While it is OFF, the DVME- 601 ignores all DPR activity except a wakeup EXEC ON command. The Executive OFF state allows the host to have random DPR memory functions (such as power-up testing) without disturbing the DVME- 601.

While the Executive is ON, it maintains a command/status area in the DPR at host address BASE + $0FFF4 through $0FFF7 (the ADSTAT and HSTSTA words). Please refer to section 3. The host must examine this bidirectional area to determine the state of the Executive. The user's host program is required to manage bit flags in this area during transfers of A/D data blocks. This will be detailed in a later section.

5.1 Executive and Monitor Interface

The Executive level is entered when the ASCII string "EXEC ON<cr>" is sent to the DVME-601 through either its DPR command area or the serial port. The Monitor level can be re-entered at any time from the serial port by pressing control-C as long as the interrupt system is intact. Back at the Monitor with the Executive ON, the DVME-601 remains Awake, but further communication with the VMEbus host is suspended and host DPR interrupts should not be sent to the DPR. The status and control state of the Executive will be preserved.

Optional User Code Debugging:With the Executive ON but back at the Monitor, optional user- written code (such as A/D math) may be executed under trace or breakpoint control. Monitor commands can now be used to investigate the current state of the DVME-601. As long as none of the critical values of the CPU registers, 68901 registers and program variables are permanently altered while at the Monitor level, the suspended state of the Executive can be re-entered using the GO command.

When the ASCII string "EXEC OFF<cr>" is sent to the DVME-601 through either the serial port or DPR, a software reset to the original power up state of the firmware will be performed.

5.2 Application Function Blocks

Commands to the Executive are written by the host in a reserved transient area near the top of the DPR. This command area contains either Application Function Blocks (AFB's) or function commands. AFB's are sequential command lists which are executed in the sequence they are written in. They are typically less than 100 bytes long. The AFB's contain addresses and parameters of internal Executive subroutines bounded by ASCII marker strings. The command list proceeds from an ASCII marker at a known starting address (BASE + $0FFEF) in descending order followed by an ASCII terminator string. Up to two AFB's

may be defined at any given time. The list of symbolic upper case ASCII function commands below the AFB's cause execution of the subroutines inside an AFB.

Program or Disk Storage of Commands

Short AFB's and function commands may be stored as constants in the user's host program controlling the 601. Longer AFB/function command lists may be stored on disk. A small user-written loader program transfers the list from disk to the DPR.

AFB definitions in the DPR start with upper case ASCII marker strings ("BEG0" or "BEG1") and end with an ASCII terminator ("ENDF"). The subroutines within the AFB's are user-defined groups of longword binary addresses of any of the built-in Executive subroutines with their binary parameters. Internal subroutines all use 4-byte longword (.L) parameters, if required. User-written subroutines which were previously downloaded to the DVME-601 can also be referenced within the AFB's. Parameters to user subroutines may be passed within the AFB list if the user writes his routines using the same parameter-passing technique. Or parameters may be passed elsewhere in the DPR.The Executive transfers the AFB's to a reserved internal local RAM area when it receives a DPR command interrupt from the host. Any function commands present in the DPR communication block are decoded at this time and appropriate control variables within the Executive are updated. Execution of the AFB's is always from the local copy of the AFB's.

Function Commands

The manner in which the AFB's execute is determined by the function commands. Within an AFB, execution proceeds in the sequential order of the subroutines. Figure 5.0 shows an example of two Application Function Blocks and a function command list loaded into the DPR by the host just before a command interrupt to the DVME-601. The right side of the figure shows the longword offset to the BASE starting address. AFB 0 contains three longword (JSR) subroutine addresses. Routine A requires two longword parameters, 0 and 1. Routine B requires no parameters. Routine C uses four parameters. AFB 1 contains only one subroutine D and it needs no parameters. Since AFB 1 is defined last (it is LOWEST in the DPR), it will be the first function executed.

Note that the internal Executive subroutines themselves find these parameters from the AFB. Coding in the routine knows how many parameters will be passed. A simple register pointer method is used. User-written routines may use the same method.

A function command list follows the two AFB's and causes execution of them after the interrupt. The function command list also uses short ASCII commands and an "ENDC" terminator. The particular command list in this example invokes automatic switching of the two AFB's (the "SWEN" command) then starts continuous running (the "CONT" command). If A/D data blocks were started with these commands, data would be placed in the DPR at high speed shortly after command start. Flag bits would be set in the ADSTAT word (and optional VMEbus interrupts) to alert the host to take data. The host may terminate execution at any time with several types of stop commands.

It is important to observe that this activity could be controlled from many host languages. The Executive DPR status bits in ADSTAT and HSTSTA (BASE + $0FFF4-7) which must be managed require bit manipulation. However, this is easily managed in C and Pascal as well as Assembly.

	Top of DPR	Offset from BASE Address
	Not available	$0FFFE-FFFF
<Hardware register>	Cmnd intrpt to DVME-601 (Wr)	$0FFFC-FFFD
<Hardware register>	VMEbus intrpt ID reg (r/w)	$0FFFA-FFFB
<Hardware register>	Enbl/Dsbl VMEbus Intrpt (bit7)	$0FFF8-FFF9
	Exec HSTSTA word (r/w)	$0FFF6-FFF7
	Exec ADSTAT word (r/w)	$0FFF4-FFF5
	Unused (available to user)	$0FFF0-FFF3
Application Function Block 0	"BEG0"	$0FFEC-FFEF
	Subr. Adrs. A	$0FFE8-FFEB
	Param. 0	$0FFE4-FFE7
	Param. 1	$0FFE0-FFE3
	Subr. Adrs. B	$0FFDC-FFDF
	Subr. Adrs. C	$0FFD8-FFDB
	Param. 0	$0FFD4-FFD7
	Param. 1	$0FFD0-FFD3
	Param. 2	$0FFCC-FFCF
	Param. 3	$0FFC8-FFCB
	"ENDF"	$0FFC4-FFC7
Application Function Block 1	"BEG1"	$0FFC0-FFC3
	Subr. Adrs. D	$0FFBC-FFBF
	"ENDF"	$0FFB8-FFBB
Function Command List	"SWEN"	$0FFB4-FFB7
	"CONT"	$0FFB0-FFB3
	"ENDC"	$0FFAC-FFAF
(Address may vary) A/D Buffer 2 Pointer	DPR commands build downward	$0FFAB
A/D Buffer 1 Pointer	A/D data blocks or S2-record downloads build upward.	$00000
	Bottom of DPR = BASE	

Data in "quotes" is upper case ASCII bytes.

A/D buffer pointers and length are user-assigned by Executive subroutines.

Figure 5.0 DPR Command Example

5.3 Application Function Block Subroutines

Executive subroutines initialize local registers and start data acquisition routines. These routines and their parameters are listed in a later section. The method of data transfer synchronization with the host is selectable (polled and/or interrupt, wait/no-wait) as well as the destination addresses of A/D data blocks. Also included are subroutines for defining the DPR single and dual buffer data transfer modes. A/D start triggering options are selectable and the timebase may be set from the AFB.

5.4 Function Block Execution

A memory write by the host on word access to DPR location BASE + $0FFFC causes a hardware interrupt to the DVME-601 local CPU. This interrupt results in either internal loading of the AFB's and/or execution of previously-loaded AFB's. If the DPR contains function commands at the time of the command interrupt, execution will take place. Therefore there are three choices for the DPR:

1. The DPR contains only AFB's. Upon command interrupt, the AFB's will be copied into internal RAM and form the definition for the next function command. No execution will occur. The Executive quickly returns to the command level.

2. The DPR contains only function commands. The command interrupt will cause execution only if an AFB was previously stored in internal RAM. If no AFB was previously loaded, a command error bit will be set in HSTSTA and the Executive will wait for the next command interrupt. The host should reset the ACK bit and command error bits.

3. The DPR contains both AFB's and function commands. The interrupt will load the AFB's and immediately cause execution. The AFB's should located at the beginnning of the DPR otherwise the function commands will atempt to execute previously loaded AFB's. If the AFB's follow certain repeating function commands, the new AFB's may never be loaded. Execution will use previously stored AFB's.

Execution Requires Function Commands:

Writing an AFB into the DPR does not execute it. Interrupting the 601 to copy the AFB internally also does not start execution. The subroutines in the AFB will only start by interrupting with a function command (at least an "STRS" followed by "ENDC").

If the Executive is ON (awake) but a control-C at the serial port has placed the DVME-601 back at the Monitor, new DPR interrupts should not be sent by the host since they will be ignored. The Monitor must be in the Go state for DPR interrupts to be processed and for AFB execution to occur. If you wish to observe AFB subroutine execution at the Monitor level, set a breakpoint within the subroutine of interest (you will need your listing file). Now start your command subroutine with a DPR interrupt. When the breakpoint hits, you will be back at the Monitor with the Exec still On but suspended. You may then single step through your code with Trace. Trace will even work with EPROM code but should not be run when local interrupts occur.

With the Executive ON and the Monitor in the GO state, the Executive sets HSTSTA DPR flag bits when it successfully accepts a command. Other flags are set after completing an A/D data transfer.

After an AFB begins execution, the DPR may be overwritten by A/D data or by a new AFB where the old AFB was originally loaded. (Do not write above Base + $FFEF.) Because of the interrupt command method, AFB's may be stopped and changed at any time.

The user is responsible for ensuring that the sequence of subroutines follows a logically correct path. A minimum amount of error checking is done to yield the highest possible throughput.

5.5 Function Command Sequences

The ASCII function commands allow an AFB to be executed once, continuously or up to 65,536 times. Two AFB's may auto-switch at high speed or simply execute both AFB's then return to the command level. Either AFB may be designated as the active block and both blocks may be cleared. Execution may be terminated at any time or an AFB may be commanded to terminate only after it finishes its current function. There is no conditional execution or branching. However, normal conditional program flow is available by writing your own local code.

5.6 Command Start-up Procedure

The following procedure is used to load commands and begin execution from the user's host program. Details of the two system variables, ADSTAT and HSTSTA will be explored in a later section. The code assumes the user is familiar with 68010 Assembly language. These are examples only and should be modified for the actual application.

1. Write the Executive ON string, "EXEC ON<cr>" into the DPR starting at location BASE + $FFE6. Note that this address differs from address BASE + $FFEC used to start AFB's or function commands. The characters must be in upper case ASCII including the space and carriage return. Do not write the <angle brackets>. The code for this is:

```
BASE    EQU     * (Put your BASE address here)
        MOVE.L  #$FFE6 + BASE, A0
        MOVE.B  #'E', (A0)+
        MOVE.B  #'X', (A0)+
        MOVE.B  #'E', (A0)+
        MOVE.B  #'C', (A0)+
        MOVE.B  #' ', (A0)+     * ASCII space
        MOVE.B  #'O', (A0)+
        MOVE.B  #'N', (A0)+
        MOVE.B  #$0D, (A0)      * carriage return
```

2. Reset the host acknowledge bit 0 of BASE+$FFF6,7. Now interrupt the DVME-601 so that it will be Awake. Remember to use word access. Any word may be used for the write.

```
        CLR.W   $FFF6+BASE
INT601: MOVE.W  #$0000,$FFFC+BASE   * Cmd intrpt to DVME-601
```

3. After a few milliseconds, verify that the Executive is ON. The DVME-601 does this by clearing all bits in the HSTSTA and ADSTAT words except ADSTAT bit 15 and HSTSTA bit 0. HSTSTA bit 9 will be set for a power up memory error.

```
ADSTAT  EQU     BASE+$FFF4
HSTSTA  EQU     BASE+$FFF6
```

*ADSTAT and HSTSTA may take a few milliseconds to be posted.
*Poll at moderate duty cycle or less.

```
        CMPI.W  #$8000,(ADSTAT)     * ADSTAT set ok?
        BNE     EXCERR              * no, Exec failed
        CMPI.W  #$0001,(HSTSTA)     * ok, check HSTSTA
        BNE     EXCERR              * no, Exec failed
```

*Both ADSTAT and HSTSTA are okay. Continue processing.

```
EXCERR: (Place code here to respond to the EXEC ON failure)
```

4. The Executive is ON. The host must reset the Host Acknowledge (HSTSTA bit 0) after each command interrupt to the DVME-601. Otherwise the Executive will lock up on the next command interrupt sent. Now write AFB's and function commands into the DPR and interrupt the DVME-601 as in the above example. The commands will overwrite the old "EXEC ON" string.

5. Check the error bits in HSTSTA to verify that the commands were accepted and are executing. The HSTSTA Acknowledge (bit 0) will be set to indicate that the Executive is processing the command. Reset it again.

6. Either poll HSTSTA and ADSTAT or respond to interrupts from the DVME-601 to transfer A/D data blocks. If the host receives an interrupt from the 601, confirm the interrupt by polling ADSTAT and HSTSTA since these bits are always set on data ready.

7. Reset ADSTAT and HSTSTA bits after taking each A/D data block. Some Executive transfer modes will wait for this host reset by polling from the DVME-601 side of the DPR. Other transfer modes will immediately start filling the next data block. The host would normally process the raw A/D data after block moving it elsewhere in system memory.

8. After the data acquisition session, stop the A/D scanning with the "STOP", "CLER", "RSET" or "EXEC OFF<cr>" function commands.

5.7 Application Function Block Syntax

A longword (4 byte, .L) communication syntax must be followed by the host when sending an AFB or function command to the DVME-601. The start of an AFB always begins with the ASCII marker string "BEG0" or "BEG1" depending on whether a new Function 0 or Function 1 is to be defined. This header longword is then followed by one or more subroutine entry addresses plus any required subroutine arguments. *All items must be in the long word format.* The AFB must be terminated with the string "ENDF". A second AFB could immediately follow or a list of function commands could follow next. If function command(s) are included, they must be terminated with the ASCII string "ENDC". An example AFB and function command is shown in Section 5.8.

The EXEC ON<cr> and EXEC OFF<cr> function commands are unique in that:

1. They always are written starting at BASE + $FFE6.

2. They always include the ASCII space character ($20) and the carriage return ($0D).

3. They do not use the "ENDC" ASCII terminator.

NOTICE

The DVME-601 is an A24 D16 VME board with a 16-bit data bus width to the VMEbus. To offer compatibility with many possible hosts, there is no connection on P2 to configure a 32-bit data bus. Using longword transfers (.L) on a 68010-based host automatically causes two 16-bit bus transfers and you may write longword code without restriction. However, 32-bit transfers on a 68020, 68030 or other 32-bit CPU in a system with 32 physical bits on the data bus require modification to create two 16-bit P1 transfers. Do not write .L instructions referencing the DVME-601 dual port RAM before doing this.

There are several ways to convert 32-bit transfers to double 16-bit transfers. You may use a system with dynamic bus sizing which recognizes the memory map reservation for the DVME-601 as a 16-bit bus area. Some assemblers allow switches to invoke automatic conversion of longword instructions to double words. Or the assembler may accept a directive identifying the processor as a 68010 even if it is actually a 68020. If all else fails, write all DVME-601 DPR references from the host as word access. Code which runs on the DVME-601's internal 68010 may use longword instructions freely.

5.8 Application Function Block and Function Command Example

The example below shows each byte in the DPR in more detail for one AFB and one function command list. Items in "quotes" are upper case ASCII characters. Notice that individual ASCII commands or markers are written in ascending memory order but the entire list is in descending order. This allows proper use of longword instructions but keeps the commands at the opposite end of the DPR away from A/D data blocks. This avoids inadvertant overwrites of A/D data blocks if the data blocks do not use the entire DPR.

Assembly language instructions illustrate the method of loading the DPR before execution.

DPR Base Address Offset	Contents	Description
$FFEF	"0"	Mark start of Function 0
$FFEE	"G"	definition.
$FFED	"E"	
$FFEC	"B"	(MOVE.L #'BEG0',$FFEC+BASE)
$FFEB	$0C	Executive subroutine address
$FFEA	$10	(MSB) 2100CH (LSB)
$FFE9	$02	
$FFE8	$00	(MOVE.L #$2100C,$FFE8+BASE)
$FFE7	$0A	Executive subroutine address
$FFE6	$14	(MSB) 2140AH (LSB)
$FFE5	$02	
$FFE4	$00	(MOVE.L #$2140A,$FFE4+BASE)
$FFE3	"F"	Mark end of Function
$FFE2	"D"	definition.
$FFE1	"N"	
$FFE0	"E"	(MOVE.L #'ENDF',$FFE0+BASE)
$FFDF	"S"	Executive Function Command.
$FFDE	"R"	[Start single execution of
$FFDD	"T"	active (NEW) function]
$FFDC	"S"	(MOVE.L #'STRS',$FFDC+BASE)
$FFDB	"C"	Mark end of function command
$FFDA	"D"	list.
$FFD9	"N"	
$FFD8	"E"	(MOVE.L #'ENDC',$FFD8+BASE)

5.9 DPR Executive Subroutines

This section lists firmware subroutines which may be accepted within an Application Function Block. After defining the AFB, a function command (described in a following section) will cause execution of the subroutine. Most subroutines deal with A/D conversion.

A symbolic ASCII name is associated with each subroutine. These names are used in the EPROM program source for customers who wish to obtain the source from DATEL.

The ASCII subroutine names should not be loaded into the DPR. The names are not 4-character function commands. Instead, a user should load the binary longword address into the AFB as shown in examples of the previous section. Use the symbolic ASCII subroutine names in EQUate tables in your host program. This will aid diagnostics in case the user contacts DATEL with software questions.

The addresses are relative to the local DVME-601 CPU. Most are calls to a jump table in the EPROM. Most subroutines require parameters. The AFB loading process must use exactly the same number of longword parameters listed in these sections. Many routines directly load registers from these parameters. If a second or Nth parameter is not needed, AFB space must still be reserved for the parameter. Use a longword of $00000000 if a parameter is not needed.

If the host controlling program inadvertantly omits a parameter, or loads a parameter that is out of range or uses an undefined subroutine address, the DVME-601 may appear to stop execution. This would be seen in the ADSTAT and HSTSTA status bits. Control can almost always be recovered by first resetting the ACK bit (HSTSTA bit 0) then executing the "CLER", "RSET" or "EXEC OFF<cr>" function commands.

The user should carefully study these subroutines since many of them perform similar functions. Internally they share common code. Subtle differences in execution deal mainly with the timing sequence of events and which events wait for other functions. Some experimentation may be needed to obtain the desired results. Many of these subroutines may also be called from local user-written code in the DVME-601. The user's subroutines must use the same parameter-passing convention as the built-in firmware subroutines.

5.9.1 A/D Conversion Start Subroutines

Refer back to register descriptions in section 4 when trying to decide which routine is suitable. The A/D command register is particularly important. Some routines use an external trigger or the Pacer timer input to start the first conversion. Then subsequent conversions are started internally in status mode. Other routines require a trigger or timer pulse for every conversion. Settling delays for high resolution inputs may use either the local Pacer timer or an external trigger.

In the routines listed below, the "Pacer timer" refers to A/D conversions started by an output from the 68901 controller jumpered on the connector to the J2-10 external trigger input. A "Pacer" subroutine also starts and stops the timer as described below. A "Trigger" subroutine does not manage the 68901 timer and instead expects an external TTL pulse.

"Status" driven conversions refer to conversions started by the local Executive subroutine by reading the A/D data register as soon as EOC goes true. This does not mean reading the data from the DPR by the host. Status conversions run as fast as the A/D conversion period plus a portion of overlapped data transfer time. Pacer timer subroutines require the PRTIMx subroutines first. PRTIMx routines load the timer values into local RAM variables. They do not load the 68901 registers directly. These registers are then loaded and started by any of the A/D subroutines in sections 5.10., 5.10.6, 5.10.9, or 5.10.11. After A/D scanning is done, the timers are stopped. PRTIMx does not have to be loaded again for the next start of A/D conversion.

If you need to load the 68901 registers directly, use the XFRBYT block transfer subroutine adjusted for a byte count of one. Transfer the byte value from an unused DPR location which was previously written from the host. The timer ouputs are squarewave and must be converted to pulses before A/D trigger usage. This occurs automatically with the J2-10 input.

5.9.2 Destination Buffers

Blocks of A/D data may be sent to the DPR or local RAM. Normally, the DPR is used as the destination. Local RAM storage (the SLLCXF routine) would only be considered for very large A/D arrays filling most of the local RAM while a previous array is being read out of the DPR. After the host reads the DPR array, and after A/D loading of the local buffer, the XLCxxx Executive subroutines allow transfer to the DPR. Both word (.W), longword (.L) and double longword (2.L's) transfers of A/D data may be used to support high level language data structures in the host.
Another reason for sending samples to local RAM first is if a user preprocessing program would be invoked after the A/D conversion and before transfer to the host.

Local RAM A/D transfers use a pointer for the intial load location. The pointer name is LOCBUF and is located in local RAM. EPROM contains a non-varying pointer to LOCBUF at $00021E. This EPROM location is a "pointer to a pointer". A/D blocks load into the address pointed by the pointer pointed to by the contents of LOCBUF. They do not load starting at LOCBUF.

A second variable CNTPTR (at $000222) is used in conjunction with LOCBUF. It contains a pointer to a data count variable. This data count variable contains the number of data "objects" (words, long words, doubles) contained in the local data buffer.

Buffer Definition

For DPR transfers, up to two buffers may be defined. A buffer definition requires both a start address and a length. The buffer addresses are offsets to the BASE address. They are not absolute addresses to either the host or the local CPU, although they must be longwords. The range is $00000000 (the base of the DPR) to $0000FFEE. The buffer length is also a longword and is added to the start address. Therefore the last usable host referenced address in a buffer = BASE + start + length.

Normally, the two buffers would not overlay each other. Nor do they have to be contiguous or have the same length. They may overwrite the DPR AFB and function command area after execution if required. Addresses higher than BASE + $FFEE should not be selected. The user is fully responsible for proper buffer definition.

One buffer is always active to receive A/D data blocks. The active buffer may be autoswitched by the SLDBUF subroutine or by a function command which selects a second AFB with the alternate buffer defined. SLDBUF switches each time another A/D subroutine is started.

The A/D routines always write to their buffers at the starting buffer address. If fewer data points are loaded than the buffer size, the unfilled portion of the buffer will be undefined. The buffers will not concantenate with each successive buffer fill (but the user could write such a subroutine).

Buffer Switching

The SLDBUF subroutine enables automatic buffer switches within each AFB whenever any of the A/D commands are invoked. The DFDBUF command must have been executed first. Using SLDBUF, the BF1ACT and BF2ACT switch commands are not required to switch buffers. BF1ACT and BF2ACT should only be used as explicit buffer steering commands from the host. SLDBUF may be combined with the XLCABF command for local RAM usage if needed. SLDBUF sets internal buffer switch-test flags and only needs to be called once at buffer setup. SLSBUF resets to single buffers only.

The SWEN Executive command is not required to switch two DPR buffers. SWEN only switches the AFB's. SLDBUF is the preferred buffer switch method.

After a buffer setup and SLDBUF AFB is executed, a single AFB as listed below will continuously write swapped buffers until an RSET or ABRT function command stops A/D sampling.

```
"BEG0"      ;AFB ASCII marker - loads at BASE+$FFEC-F
$000---XX   ;any longword A/D subroutine address with implied swap
"ENDF"      ;marks the end of AFB 0
"CONT"      ;makes the AFB loop continuously
"ENDC"      ;marks the end of function commands.
```

DSP Applications, DPR Buffer timing, etc.

For long-running DSP/FFT and spectral applications where a precision unbroken A/D start clock is required, you must examine the total number of samples to be taken. If less than approximately 30,000 samples are needed, local RAM may be filled. For more than 30K samples without missing a high speed trigger, the fast buffer switch method (see below) is needed.

Using the SLDBUF routine, be aware that software buffer switch time is required through the Executive using the swapped dual buffer method as each AFB is completed. (This switch time occurs AFTER each multisample A/D subroutine or AFTER the AFB is done and not between samples). This switch time varies with many parameters including the data object size, A/D trigger and data transfer method, etc. Many other Executive functions are performed during this time including serial port ring buffer management, etc. A rough guide is 140 -300 microseconds to load and test pointers, post DPR bits, etc. Exact timing should be empirically tested.

Fast Buffer Switch Method

The 601's Executive swapped buffer functions trade off ease of programming in exchange for this timing delay. If you wish to avoid this management overhead, refer to the downloaded example program AD2BUF.ASM on disk for a much faster buffer switch method which allows non-stop A/D triggering and simultaneous DPR transfer.

5.9.3 Host Data Transfer Synchronization

Flag bits in the ADSTAT word identify the active buffer and whether it contains data ready for host transfer. If enabled, VMEbus interrupts will be sent when the data ready flags are updated by the Executive. The Executive has two options (wait and no-wait) for synchronization with the host. The wait mode guarantees no loss of data but slaves A/D scan timing to the host. The no-wait mode is useful for "seamless" recording such as DSP applications where the A/D start clock should not stop or have timing jitter. These options are as follows:

1. The Executive will transfer A/D data blocks to the DPR and will wait until the host resets the ADSTAT bit flag corresponding the last active buffer. The Executive polls the appropriate bit. The host transfers the data then resets the bit. ADSTAT bit 1 is set by the Executive to indicate buffer 1 has valid data. ADSTAT bit 2 is used for buffer 2.

2. The Executive will not wait for the ADSTAT data ready bits to be reset. New A/D data will immediately overwrite the old data as A/D conversions are made. If the host does not reset the bits, it will not detect when new data is ready in the buffer since the Executive does not reset these bits in either the wait or no-wait modes.

This second "no-wait" mode should only be used for swapped double-buffer modes. This will allow the host a reasonable amount of time to respond to the data ready condition. Although the host can do the actual memory block transfer much faster than A/D filling, the response to the bit polling or interrupt arbitration may make this timing close. The user should verify this with tests and an oscilloscope.

The user's host program should only reset the flag bits. Other status bits should be left alone or read only. The Executive keeps an internal copy of both ADSTAT and HSTSTA to avoid disruption by the host.

The ADSTAT data ready bits 1 and 2 will send a VMEbus interrupt to the host if the interrupt enable register at BASE + $FFF8, bit 7 is set to 1 on word access.

5.10 Subroutine List

A/D start routines may be either one single channel conversion, "N" single channel conversions using a fixed channel address, one multichannel sequential scan, or "N" scans. The word "scan" in the routines below means a multichannel sequential scan with the DVME-601 automatically controlling the channel addressing and incrementing the channel address register after each conversion. Scans must always have the start and final address registers initialized by the host program. Subroutine addresses and parameters must be in hexadecimal.

The subroutines are listed in the normal sequence of the host program. Buffers should be defined first, followed by channel addresses, and timers. An optional program download should be done then the conversion session. The Executive subroutines are divided into the following major categories:

<u>General purpose</u>

5.10.1: Define buffer destination addresses of A/D data blocks and host synchronization.

5.10.2: A/D channel address registers and timer registers initialization.

<u>Multichannel sequential scan routines</u>

5.10.3: A/D standard scan start routines.

5.10.4: A/D Pacer timer/data read scan start routines.

5.10.5: A/D trigger/data read scan start routines.

5.10.6: A/D scan start with Pacer timer settling delay.

5.10.7: A/D scan start with trigger settling delay.

<u>Single channel conversion routines</u>

5.10.8: A/D standard single channel start routines.

5.10.9: A/D Pacer timer/data read single channel start routines.

5.10.10: A/D trigger/data read single channel start routines.

5.10.11: A/D single channel start with Pacer timer settling delay.

5.10.12: A/D single channel start with trigger start.

DVME-601 EPROM A/D Subroutines

		Number of Channels per Subroutine					
		Single Channel		One Multichannel Scan (8)		"N" Multichannel Scans (8)	
	A/D to RAM Data Transfer Method	Polled EOC Status	Fast Thruput	Polled EOC Status	Fast Thruput	Polled EOC Status	Fast Thruput
A/D Start Source	Write $CXXXX (1st sample) (8)	STSNSN (1) STMLSN (2)	FASNSN (1) FAMLSN (2)	STSNSC	FASNSC	STMLSC	FAMLSC
	External Trigger (9)	TMRDSN (3) TRDLSN (5) TMDLSN (6)		TRRDSC (4) TRDLSC (5)		TMRDSC (10) TMDLSC (5)	
	Pacer Timer (68901) (7)	PMRDSN (3) PCDLSN (5) PMDLSN (6)		PCRDSC (4) PDCLSC (5)		PMRDSC (4) PMDLSC (5)	

1. One sample
2. "N" samples
3. One trigger per "N" samples
4. One trigger per scan
5. One trigger per sample
6. "N" triggers for "N" samples
7. Subroutine starts/stops timer (not recommended for most applications)
8. Multisample routines start the A/D by reading the A/D data register after writing to $CXXXX to start the first sample.
9. Use the external trigger routines for timer-based looping or swapped buffer sampling which runs continuously. Start the timer first with XFRBYT directly to the 68901 registers. Do not use the PRTIMx subroutines with the external trigger routines.
10. One trigger per "N" scans.

The Pacer clock must be externally connected for use with the trigger input. Connect J2-19 to J2-10.

5.10.1 Define Buffers and Host Synchronization

[Use longword format for all addresses and parameters]

Subroutine Name	Local entry address	Description and call sequence
DFSBUF	$0012E	Define a single DPR buffer. Calling syntax: $0012E - Subroutine entry address XXXXX - Buffer start address (0 = DPR base) XXXXX - Buffer length (max = 65K, 16-bit words)
DFDBUF	$00134	Define two DPR buffers. Calling syntax: $00134 - Subroutine entry address XXXXX - Buffer 1 start address XXXXX - Buffer 1 length (max = 65K, 16-bit words) XXXXX - Buffer 2 start address XXXXX - Buffer 2 length (max = 65K, 16-bit words) For DFSBUF and DFDBUF, the bottom of the DPR is defined as zero
SLSBUF	$0013A	Select single DPR buffer data transfers. Calling syntax: $0013A - Subroutine entry address
SLDBUF	$00140	Select double DPR buffer data transfers. Calling syntax: $00140 - Subroutine entry address
BF1ACT	$001E8	Make DPR Buffer #1 the active buffer (default). Calling syntax: $001E8 - Subroutine entry address
BF2ACT	$001EE	Make DPR Buffer #2 the active buffer. Calling syntax: $001EE - Subroutine entry address
	Wait/No-Wait Transfer Modes	
SLOVWR	$00146	Select data transfer to overwrite the DPR buffer and ignore data ready flags. This is the default mode. Calling syntax: $00146 - Subroutine entry address
SLWTEM	$0014C	Select DPR data transfer to wait for host reset of data ready flags. Calling syntax: $0014C - Subroutine entry address

Select Local RAM or DPR Transfers

SLLCXF	$00158	Select A/D data transfer to local buffer area. Calling syntax: *$00158 - Subroutine entry address* Note: No data ready bits are set in ADSTAT.
SLDPXF	$0015E	Select A/D data transfer direct to DPR buffer. This is the default mode. Calling syntax: $0015E - Subroutine entry address
INBFPT	$00152	Initialize or change the local A/D data buffer pointer for local user programs. Calling syntax: $00152 - Subroutine entry address XXXXX - Local buffer new starting address (absolute address in local memory).

Start Local RAM-to-DPR Transfer

XLCBF1	$00164	Transfer data from local A/D buffer to previously defined DPR buffer #1. Calling syntax: $00164 - Subroutine entry address $00000 - For WORD transfers or $00001 - For LONG WORD transfers or $00002 - For DOUBLE transfers
XLCBF2	$0016A	Transfer data from local A/D buffer to previously defined DPR buffer #2. Calling syntax: $0016A - Subroutine entry address $00000 - For WORD transfers or $00001 - For LONG WORD transfers or $00002 - For DOUBLE transfers
XLCABF	$00170	Transfer data from local A/D buffer to previously defined and currently active DPR buffer. Calling syntax: $00170 - Subroutine entry address $00000 - For WORD transfers or $00001 - For LONG WORD transfers or $00002 - For DOUBLE transfers
XLCDPR	$00176	Transfer data from local A/D buffer to DPR starting at the address offset in the passed argument. Calling syntax: $00176 - Subroutine entry address $00000 - For WORD transfers or $00001 - For LONG WORD transfers or $00002 - For DOUBLE transfers XXXXX - Offset from DPR BASE address

5.10.2 A/D Channel Address Register And Timer Initialization

The following routines write directly to the A/D channel address registers or the RAM variables for the 68901 controller registers. The 68901 registers are not directly written until an A/D command using the timer. Do not use the PRTIMx routines for general purpose non-A/D timer control. Use XFRBYT instead.

Subroutine Name	Local entry address	Description and call sequence
LDADRG	$00128	Load start and final channel address registers. Calling syntax: $00128 - Subroutine entry address XXXXX - Start channel address XXXXX - Final channel address
PRTIMA	$001D6	Initialize Timer A control variables. Calling syntax: $001D6 - Subroutine entry address XXXXX - Timer A data register XXXXX - Timer A control register
PRTIMB	$001DC	Initialize Timer B control variables. Calling syntax: $001DC - Subroutine entry address XXXXX - Timer B data register XXXXX - Timer B control register
PRTIMC	$001E2	Initialize Timer C control variables. Calling syntax: $001E2 - Subroutine entry address XXXXX - Timer C data register XXXXX - Timer C control register

5.10.3 A/D Standard Scan Start Routines

Subroutine Name	Local entry address	Description and call sequence
FASNSC	$0017C	Perform single "fast-throughput" A/D channel scan. Calling syntax: $0017C - Subroutine entry address XXXXX - Start channel address XXXXX - Final channel address (Note: Fast thruput may have insufficient set-tling times for multiple channels. Single channel is recommended).
STSNSC	$00188	Perform single "status-driven" A/D channel scan. Calling syntax: $00188 - Subroutine entry address XXXXX - Start channel address XXXXX - Final channel address

FAMLSC	$00182	Perform "N" "fast-throughput" A/D channel scans. Calling syntax: $00182 - Subroutine entry address XXXXX - Start channel address XXXXX - Final channel address XXXXX - Desired number of scans
STMLSC	$0018E	Perform "N" "status-driven" A/D channel scans. Calling syntax: $0018E - Subroutine entry address XXXXX - Start channel address XXXXX - Final channel address XXXXX - Desired number of scans

5.10.4 A/D Pacer Timer/Data Read Scan Start Routines

Subroutine Name	Local entry address	Description and call sequence
PCRDSC	$00194	Perform single "Pacer-driven" A/D channel scan. The first conversion is started by the Pacer. A/D conversions for the rest of the scan are started by reading the A/D data. Calling syntax: $00194 - Subroutine entry address XXXXX - Start channel address XXXXX - Final channel address
PMRDSC	$0019A	Perform "N" "Pacer-driven" A/D channel scans. The first conversion is started by the Pacer. A/D conversions for the rest of the scans are started by reading the A/D data. Calling syntax: $0019A - Subroutine entry address XXXXX - Start channel address XXXXX - Final channel address XXXXX - Desired number of scans

5.10.5 A/D Trigger/Data Read Scan Start Routines

Subroutine Name	Local entry address	Description and call sequence
TRRDSC	$001A0	Perform single A/D scan started by external trigger. The first conversion is started by the trigger. Conversions for the rest of the scan are started by reading the A/D data. Calling syntax: $001A0 - Subroutine entry address XXXXX - Start channel address XXXXX - Final channel address

TMRDSC	$001A6	Perform "N" A/D scans started by external trigger. The first conversion is started by the trigger. Conversions for rest of the scans are started by reading the A/D data. Calling syntax: $001A6 - Subroutine entry address XXXXX - Start channel address XXXXX - Final channel address XXXXX - Desired number of scans

5.10.6 A/D Scan Start With Pacer Timer Settling Delay

Subroutine Name	Local entry address	Description and call sequence
PCDLSC	$001AC	Perform single A/D scan incorporating Pacer generated delay between channel select and start of conversion. Timer(s) must have been previously programmed. Calling syntax: $001AC - Subroutine entry address XXXXX - Start channel address XXXXX - Final channel address
PMDLSC	$001F4	Perform "N" A/D scans using Pacer generated delay between channel select and start of conversion. Timer(s) must have been previously programmed. Calling syntax: $001F4 - Subroutine entry address XXXXX - Start channel address XXXXX - Final channel address XXXXX - Desired number of scans

5.10.7 A/D Scan Start With External Trigger Settling Delay

Subroutine Name	Local entry address	Description and call sequence
TRDLSC	$001FA	Perform a single A/D scan using an external trigger-generated delay between channel select and the start of conversion. Calling syntax: $001FA - Subroutine entry address XXXXX - Start channel address XXXXX - Final channel address

TMDLSC	$00200	Perform "N" A/D scans using an external trigger-generated delay between channel select and the start of conversion. Calling syntax: $00200 - Subroutine entry address XXXXX - Start channel address XXXXX - Final channel address XXXXX - Desired number of scans

5.10.8 A/D Standard Single Channel Start Routines

Subroutine Name	Local entry address	Description and call sequence
FASNSN	$001B2	Perform single channel "fast-throughput" A/D conversion. Calling Syntax: $001B2 - Subroutine entry address XXXXX - Channel address
STSNSN	$001BE	Perform single channel "status-driven" A/D conversion. Calling Syntax: $001BE - Subroutine entry address XXXXX - Channel address
FAMLSN	$001B8	Perform "N" "fast-throughput" single channel A/D conversions. Calling syntax: $001B8 - Subroutine entry address XXXXX - Channel address XXXXX - Desired number of conversions
STMLSN	$001C4	Perform "N" "status-driven" single channel A/D conversions. Calling syntax: $001C4 - Subroutine entry address XXXXX - Channel address XXXXX - Desired number of conversions

5.10.9 A/D Pacer Timer/data Read Single Channel Start Routines

Subroutine Name	Local entry address	Description and call sequence
PMRDSN	$001CA	Perform "N" single channel A/D conversions. The first conversion is started by the Pacer. Remaining conversions are started by reading the A/D data. Calling syntax: $001CA - Subroutine entry address XXXXX - Channel address XXXXX - Desired number of conversions

5.10.10 A/D Trigger/Data Read Single Channel Start Routines

Subroutine Name	Local entry address	Description and call sequence
TMRDSN	$001D0	Perform "N" single channel A/D conversions. The first conversion is started by the external trigger. Remaining conversions are started by reading the A/D data. Calling syntax: $001D0 - Subroutine entry address XXXXX - Channel address XXXXX - Desired number of conversions

5.10.11 A/D Single Channel Start with Pacer Timer Settling Delay

Subroutine Name	Local entry address	Description and call sequence
PCDLSN	$00206	Perform single A/D channel conversion using a Pacer timer-generated delay between channel select and the start of conversion. Timer(s) must have been previously programmed. Calling syntax: $00206 - Subroutine entry address XXXXX - Channel address
PMDLSN	$0020C	Perform "N" single A/D conversions using a Pacer timer-generated delay between channel select and the start of conversion. Timer(s) must have been previously programmed. Calling syntax: $0020C - Subroutine entry address XXXXX - Channel address XXXXX - Desired number of conversions

5.10.12 A/D Single Channel Trigger Start

Subroutine Name	Local entry address	Description and call sequence
TRDLSN	$00212	Perform single channel A/D conversion started by external trigger. Calling syntax: $00212 - Subroutine entry address XXXXX - Channel address This subroutine waits for the trigger and produces one sample.

TMDLSN	$00218	Perform "N" single channel A/D conversions. ALL conversions are started by the external trigger. A stable trigger source should be used. Calling syntax: $00218 - Subroutine entry address XXXXX - Channel address XXXXX - Desired number of conversions

Memory Block Transfers

Three memory block transfer subroutines are included which move bytes, words or longwords. These transfer instructions copy from anywhere to anywhere between DPR, local RAM and EPROM. All three take source address, destination address and object count longword binary parameters. ALL ADDRESSES are relative to the LOCAL CPU. By adjusting the object count, single bytes, words or longwords may be transferred.

Use CAUTION to avoid overwriting the local system parameters or exceeding the range of transfers. Use even addresses only for word and longword transfers. The source array remains in memory after the transfer. The XFRBYT memory transfer allows full control of the 68901 peripheral registers from the DPR. These transfer subroutines may also be used as a downloader for fast absolute binary executable code, overlays or tables instead of the DPDL subroutine. S2 records will not be required, however no checksum error-checking computation is done. Block sizes are less than half the equivalent S records and will copy faster. Overlapped transfers (sharing source and destination memory) will succeed for shifting a block DOWN (lower memory) since the pointers autoincrement. Users should examine the EPROM code (or download their own small block mover into high local RAM) to autodecrement for upshifted overlapped block transfers. This will avoid overwriting source that is not yet transferred.

The three memory transfer EPROM subroutine hexadecimal jump table addresses and their DPR AFB parameters are as follows:

XFRBYT Adrs=$22E	XFRWRD Adrs=$234	XFRLNG Adrs=$23A
Src.Adrs. xxx.L	Src.Adrs. xxx.L	Src.Adrs. xxx.L
Dest.Adrs. xxx.L	Dest.Adrs. xxx.L	Dest.Adrs. xxx.L
Byte Cnt. xxx.L	Word Cnt. xxx.L	Long Cnt. xxx.L

The following example moves a non-overlapped block within the DPR

```
"BEG0"      ;ASCII marker (omit the quotes). Load at BASE+$FFEC
00000234    ;XFRWRD subroutine longword address
00040000    ;source=bottom of DPR longword address
00041000    ;destination=4096 bytes higher in DPR
00000100    ;256 "objects" (words) moved
"ENDF"      ;ASCII markers
"STRS"      ;This function command executes the AFB when interrupted
"ENDC"
```

Function Control Words and Command List

The following function commands must be entered in the DPR as upper case ASCII longwords. They are not entered as binary addresses like the Executive subroutines. The list below is divided into ASCII markers, AFB execution commands, a DPR download command (DPDL) and resets. The ACK bit (HSTSTA bit 0) must be reset to zero by the host before executing any of these commands. Failure to do this after each command will lock up further commands.

The EXEC ON<cr> and EXEC OFF<cr> are unique in that they require the carriage return character ($0D) and must be loaded starting at BASE + FFE6. They do not require the ENDC terminator. Both EXEC OFF<cr> and EXEC ON<cr> require reset of the ACK bit before the command interrupt.

The REP0 and REP1 function commands require a longword parameter of 65,536 ($0000FFFF) repetitions maximum. All other function commands do not use parameters.

Command Name	Description
EXEC ON<cr>	Place DVME-601 at the Executive level and in the Awake mode. The carriage return and space ASCII characters are required. The load address must start at BASE + FFE6. Do not run two or more EXEC ON commands in sequence. Undefined operation may result. To restore the system, do an EXEC OFF/ EXEC ON cycle or hardware reset.
EXEC OFF<cr>	Place DVME-601 at the Monitor level and in the Asleep mode. The previous Executive state will be lost but DPR data will not be changed. The carriage return and space ASCII characters are required. The load address must start at BASE + $FFE6.
	ASCII Markers
BEG0	Beginning of Application Function Block 0 definition. Subroutine addresses follow.
BEG1	Beginning of Application Function Block 1 definition. Subroutine addresses follow.
ENDF	Mark end of Application Function Block definition.
ENDC	Mark end of function command list. One or more of the Execution function commands listed below must precede ENDC.
	Resets
RSET	Executive Software Reset - Aborts and clears all AFB's. Restores original Executive start-up state. RSET does not produce an ACKnowledge bit in the DPR status words.
ABRT	Abort any function in progress. Does not clear AFB's.
CLER	Clears all defined AFB's after current function finishes execution.
	Execution commands
STRS	Execute currently active AFB once then return to command level.
STRD	Execute currently active AFB once then immediately execute the alternate AFB, if defined. Return to the command level when done.
SEL0	Make AFB 0 the active function block.
SEL1	Make AFB 1 the active function block.

REP0 — Load "N" repeat count for AFB 0. The default count is one if REP0 is not sent. The maximum count is 65,536 ($0000FFFF) and must be a longword.

REP1 — Load "N" repeat count for AFB 1. The default count is one if REP1 is not sent. The maximum count is 65,536 ($0000FFFF) and must be a longword.

SWEN — Enable AFB auto-switch mode. Both AFB's must have been defined. Use CONT or STRD to start execution. SWEN switches AFB's, not buffers. Buffers will switch only if SLDBUF or BF1ACT or BF2ACT are switched in the two AFB's.

SWDS — Disable AFB auto-switch mode.

CONT — Begin continuous execution of previously-defined AFB's. If only one AFB is defined, it will repeat indefinitely after execution of all internal subroutines. STOP, ABRT, RSET, or EXEC OFF must be used to halt execution. About 300 microseconds Executive overhead occurs with each pass through an AFB "loop". Write downloaded code if higher speed is needed.

Note: AFB plus function command list length must be 255 bytes maximum. Use downloaded subroutines for complex functions.

STOP — Disable continuous execution. The AFB in process will stop immediately. AFB definitions will be retained.

DPR Download

DPDL — Download ASCII S2-record subroutine module through the Dual Port RAM. The S2 record must have been previously loaded in the DPR starting at BASE + 0. The S2 record must contain valid load addresses relative to the local DVME-601 CPU. Execution of the subroutines does not automatically start and must be commanded with another AFB definition and load after successful download. Checksum errors set bit 1 of HSTSTA and must be reset by the host program. The first record should be type S0 and the last record should be type S8 or S9.

Note: Download of user-written executable code is only necessary if you need enhanced operation beyond that offered by the Executive subroutines. You may download either S2 ASCII records using DPDL or download absolute binary executable image using the XFRxxx block transfers. You may also download S2 records through the auxiliary serial port using the Monitor SRDL command.

Important: The S2 records must contain exactly ONE control character (either CR or LF) between records.

Use the reset functions to abort a download if it does not indicate completion in a reasonable time. Poll HSTSTA bit +1.

Downloaded programs must use the DVME-601 parameter-passing conventions to access local EPROM subroutines or Executive functions. Obtain the DVME-601 source files before proceeding.

5.11 Command/Status Control Word Definitions

This section describes the two software control words in the DPR. The words are ADSTAT (A D status) at DPR BASE + $FFF4,5 and HSTSTA (Host status) at DPR BASE + $FFF6,7. These words contain flag bits which indicate the status of the Executive. Most of the bits should only be read by the host program but a few, such as HSTSTA ACK and the ADSTAT data ready bits, must be periodically reset by the host. Your host program must be capable of bit manipulation.

The Executive keeps a copy of these control words in internal system RAM (TADSTAT and THSTSTA) to avoid disruption by the host. Both the DPR word and its internal copy will coincide most of the time. The Executive also maintains other internal RAM control words which would only be of interest for local user programs. Both the DPR versions of HSTSTA and ADSTAT are defined only when the Executive is ON.

ADSTAT and HSTSTA are maintained only by the EXEC software. They are not hardware registers. If the EXEC is not used, these words have no meaning.

At power up, these control words are undefined since the Executive is in the OFF state and the DVME-601 does not make access to the DPR. After the Executive has been interrupted ON, HSTSTA contains $0001 and ADSTAT contains $8000. These status words acknowledge that the Executive is now ON and no other operations should be attempted until these words are present. The ACK bit (HSTSTA bit 0) must be reset before any further operations.

On the remote possibility that ADSTAT and HSTSTA contain $8000 and $0001 at power-up by chance, the host should write and read a different code before interrupting the Executive. A full-blown walking 0's and 1's test of the entire DPR by the host is suggested since the DVME-601 does not test the DPR at power-up.

When transitioning from the Executive ON to OFF state, first reset the ACK bit then command EXEC OFF<cr>. The Executive will write an epitaph of ADSTAT = $0000 and HSTSTA = $0001 which your program should verify. Undefined bits in the following definitions may be any state. See the ACK protocol discussion in a section below.

HSTSTA-High Byte
Address: DPR BASE + $FFF6,7

15	14	13	12	11	10	9	8
Func 1 Active Flag	Func 0 Active Flag	Func 1 Def. Flag	Func 0 Def. Flag	Contrl S Flag	Not used	Not used	Dwn Lod Comp. Flag

Bit 8 - Subroutine Module Down Load Complete Flag (for DPL command only)
- 0 - Module down load not complete
- 1 - Module down load has been completed

[Use EXEC OFF if the download does not complete in a reasonable time.]

Bit 9 - Power Up Memory Failure
- 0 - No error
- 1 - Memory error Will attempt normal operation

Bit 10 - Not used

Bit 11 - Control-S Received on Serial Port
- 0 - Control S not received
- 1 - Control S received - waiting for Control-Q (other processing will be suspended)

Bit 12 - Function 0 (AFB 0) Defined Flag
- 0 - Function 0 is not defined
- 1 - Function 0 is defined

Bit 13 - Function 1 (AFB 1) Defined Flag
0 - Function 1 is not defined
1 - Function 1 is defined
Bit 14 - Function 0 (AFB 0) Active Flag
0 - Function 0 not active
1 - Function 0 is active
Bit 15 - Function 1 (AFB 1) Active Flag
0 - Function 1 not active
1 - Function 1 is active

HSTSTA - Low Byte
Address: DPR BASE + $FFF6,7

7	6	5	4	3	2	1	0
Func 1 Comp. Flag	Func 0 Comp. Flag	Func 1 Error	Func 0 Error	Invalid Command Error	Func Command Error	Dwn Lod Command Error	601 to Host Ack

Bit 0 - DVME-601 to VME Bus Host Acknowledge Flag
0 - Idle or still being transferred to local RAM command area. Not processing yet.
1 - Function(s) and/or Command Acknowledge (command being processed but not done nor error-free)

[ACK must be reset by the host before interrupting the Executive.] Poll ACK at moderate duty cycle and allow sufficient time for list transfer and execution setup.

Bit 1 - Module Down Load Command Error Flag
(for DPL command only)
0 - No Error Detected
1 - Down Load Command Error
Bit 2 - Function Command Error Flag
0 - No Error Detected
1 - Function Command Error
Bit 3 - Invalid or Unrecognized Command Error Flag
0 - No Error Detected
1 - Invalid or Unrecognized Command
Bit 4 - AFB 0 Function Error Flag
0 - No Error Detected
1 - Function 0 Definition or Operational Error
Bit 5 - AFB 1 Function Error Flag
0 - No Error Detected
1 - Function 1 Definition or Operational Error
Bit 6 - Function 0 Process Complete Flag
0 - Function 0 idle or still processing
1 - Function 0 processing complete
Bit 7 - Function 1 Process Complete Flag
0 - Function 1 idle or still processing
1 - Function 1 processing complete

Bits 6 and 7 are set as AFB's are swapped. They must be read then reset by the host so they may be detected after each function is done. Bits 6 and 7 refer to AFB's only. They are not used for function commands.

The "function done" bits 6 and 7 in HSTSTA (BASE + $FFF7) indicate completion of an AFB, not a function command. Bit 6 would be set for example with the following AFB:

```
BEG0
—
—
ENDF
STRS
—
—
ENDC
```

The bit is set as the CPU executes the "STRS" command.

If the command list has no AFB, the function done bits would not be set, even though the CPU executed a series of commands:

```
DPDL
ENDC
```

Remember to reset these bits from the host in order to detect them the next time. See page 5-24.

ADSTAT - High Byte
Address: DPR BASE + $FFF4,5

15	14	13	12	11	10	9	8
Active Oper Level		Pacer Clk Ini Flag		DPR Xfr Wait Enable	A/D to Loc Buf Select		DPR Xfr Mod Select

Bit 8 - Dual Port Ram data Transfer Mode Select
- 0 - Single buffer transfer mode selected
- 1 - Double buffer transfer mode selected

Bit 9 - Not used

Bit 10 - A/D data to Local Buffer Transfer Enable
- 0 - Transfer A/D data directly to Dual Port RAM
- 1 - Transfer A/D data to Local Buffer area

Bit 11 - DPR Transfer Wait for Host Enable*
- 0 - Over write data in Dual Port RAM
- 1 - Wait for Host to Reset Data Ready Flag

Bit 12 - Not used

Bit 13 - Pacer Clock Initialization Flag
(set by PRTIMx subroutines)
- 0 - Pacer Clock has not been defined
- 1 - Pacer Clock has been defined

Bit 14 - Not used

Bit 15 - Current Active Operational Level
(this bit is fully managed by the Executive)
- 0 - At MONITOR Level
- 1 - At EXECUTIVE Level

* Bit 11 is read-only by the host. Use SLOVWR and SLWTEM to control the mode.

ADSTAT - Low Byte
Address: DPR BASE + $FFF4,5

7	6	5	4	3	2	1	0
EOS Status Flag	EOC Status Flag	Not Used	Buff 2 Defined Flag	Buff 1 Defined Flag	Buff 2 Data Rdy Flg	Buff 1 Data Rdy Flg	Active Xfr Buf Flag

Note: Bits 1 and 2 are not set if data is steered to local RAM.

Bit 0 - Dual Port RAM Active Transfer Buffer Flag
0 - Buffer #1 is active
1 - Buffer #2 is active
Bit 1 - DPR Buffer #1 Data Ready Flag*
0 - DPR Buffer #1 Empty
1 - DPR Buffer #1 Data Ready
[Setting bit 1 by the Exec sends a VMEbus interrupt if enabled. Bit 1 must be reset by the host after data transfer if the wait-for-host mode is enabled.]

Bit 2 - DPR Buffer #2 Data Ready Flag*
0 - DPR Buffer #2 Empty
1 - DPR Buffer #2 Data Ready
[Setting bit 2 by the Exec sends a VMEbus interrupt if enabled. Bit 2 must be reset by the host after data transfer if the wait-for-host mode is enabled.]

Bit 3 - DPR Buffer #1 Defined Flag
0 - DPR Buffer #1 Not Defined
1 - DPR Buffer #1 Defined
Bit 4 - DPR Buffer #2 Defined Flag
0 - DPR Buffer #2 Not Defined
1 - DPR Buffer #2 Defined
[Bits 3 and 4 should be confirmed after defining the buffers].

Bit 5 - Not used
Bit 6 - A/D End of Conversion (EOC) Status Flag
0 - Not at End of Conversion
1 - End of Conversion
Bit 7 - A/D End of Scan (EOS) Status Flag
0 - Not at End of Channel Scan
1 - End of Channel Scan

[EOS is set when the Executive detects that the current and final channel address registers are equal. EOC and EOS are not hardware mapped in ADSTAT].

* These bits are not set if data is sent to local RAM.

5.12 DPR ACKnowledge Command Protocol

The following flowchart should be used for executing all commands from the DPR:

1. Reset the ACKnowledge status bit to zero in HSTSTA bit 0. This will allow immediate execution of the next command interrupt to the DVME-601. The reset will also let the host program detect when the Executive has started processing a command. Failure to reset the ACK bit before the command interrupt to the DVME-601 will cause the Executive to suspend processing while it polls the ACK bit waiting for a zero state. Further interrupts to the DVME-601 would nest interrupts and perhaps cause a general lockup.

2. Load the DPR with the ASCII AFB, subroutine addresses, optional longword parameters, function commands and ASCII markers.

3. Interrupt the Executive to process the command. The Executive will rapidly set the ACK bit. This indicates that the command has been loaded internally and is about to be processed. The ACK bit does not indicate that the command is done or can even be processed correctly. The ACK will remain set to 1 until reset by the host.

4. Poll the ACK bit. If the Executive has not set the ACK bit to 1 after a few microseconds, the DVME-601 has not started to process the command. An EXEC OFF should be attempted or bus reset to regain control.

5. If the ACK bit is set to 1, poll HSTSTA bits 1 through 5 to detect any errors. If these bits are set, branch to code in the host to process these errors. Perhaps the command load was incorrect, a wrong subroutine address was used, an incorrrect number of parameters, byte or word parameters or parameters out of range. The host program must reset these error bits in order to detect them the next time. Reset the ACK bit at this time also.

6. To retrieve A/D data, either use bit polling or VMEbus interrupts. Poll the the data ready bits (ADSTAT bits 1 or 2), depending on which buffer is active. When either bit is set to 1, A/D data may be transferred out of the DPR from the BASE offset address choosen for that buffer. The host must reset these bits after the transfer. Note that some subroutines will suspend A/D scanning until these bits are reset by the host.

If interrupts to the host were enabled, setting data ready ADSTAT bits 1 and 2 by the Executive will send a VMEbus interrupt. The host program should poll these bits to confirm the source of the interrupt. ADSTAT 1 and 2 are the only Executive bits which are related to interrupts. A user's local program could also send VMEbus interrupts but without affecting these bits unless an Executive routine was called locally.

Command Hints

Overwriting Old Commands

To prevent confusion by the command transfer function, load a zero longword ($00000000) below the "ENDF" if a function command or second AFB is not also loaded in the same sequence. The 601 continues the decrementing longword parse if it encounters an "ENDF". After an "ENDF", it stops the parse at either the "ENDC" or a longword which doesn't compare to any other function command, such as $00000000. While transferring the AFB, the 601 is looking for the "ENDF". Do not accidentally use the same binary longword pattern as a parameter for user-written subroutines.

Programming in C and Other High Level Languages

Users programming the 601 only in C should understand pointer and memory read/write operations thoroughly. Programmers should have a background in microcomputers and hardware peripherals. The VMEbus and 68XXX CPU's have their own idiosyncracies here too for byte, word, and longword operations. Some compilers are fussy about variable assignments at the time of pointer declarations. Some pointer operations behave strangely with ASCII string assignments. Pointer arithmetic (increment, decrement and adding offsets) may have anomolies. Remember that pointer is really an address which is a 4-byte unsigned integer for 68XXX CPU's. Beware of signed and unsigned, short and long declarations, statics and automatics. Use even alignment for words and longwords.

If you are unsure of 68000 longword addressing for your compiler, examine the assembly output of your compiler very carefully. If it is too difficult to follow and does not function properly, consider coding sections in assembly and get expert help.

If there is any doubt about how your high level language loads 601 commands into the DPR, single step your host program and stop it just before your host writes a command interrupt to the 601. Control-C back to the 601's Monitor and examine the DPR from the 601 side. Look at the hex and ASCII display using a Monitor command such as MD 4FFC0,100. The ASCII longword character strings are in ascending order but the entire command list descends in 4-byte longwords. Examine Section 5.8 in the manual.

Command Example

The following is a typical command list. Where symbolic names are shown or $hexadecimal parameters, you should use 32-bit binary longwords. ASCII 32-bit markers are indicated by "quotes" (don't load the quote symbols). Comments are listed after each longword.

After doing EXEC ON, this command example performs swapped buffer A/D scans in the wait mode requiring ready bit reset by the host after each buffer fill. Notice that AFB 0 is used twice. After it is executed the first time to define buffers, the AFB is no longer needed and may be overwritten. The second "CONT" AFB runs indefinitely and must be stopped with an RSET/ENDC function command. All commands load at BASE + $FFEF and downward. Connect the Timer A output to the A/D trigger input J2-10 to 17.

```
"BEGO"      ;ASCII marker for the start of AFB 0
SLWTEM      ;wait mode. This is subroutine address $0014C.
DFDBUF      ;define double buffer addresses (subroutine $0134)
$00000000   ;Buf 1 start=BASE + 0 (bottom of DPR)
$00000010   ;Buf 1 length=32-bytes (16 A/D samples in word mode)
$00000020   ;Buf 2 start=BASE + $20
$00000010   ;Buf 2 length=32
SLDBUF      ;select double buffer transfers (enables autoswitch)
PRTIMA      ;setup Timer A
$00000004   ;data register=about 13 microseconds per trigger
$00000001   ;control register=delay mode (squarewave)
"ENDF"      ;ASCII terminator for AFB 0
"STRS"      ;function command to execute AFB 0 once then stop
"ENDC"      ;ASCII terminator
```

Reset the HSTSTA ACK bit, interrupt the 601, then moderately poll the ACK bit. When set, check the HSTSTA and ADSTAT error and status bits. Now overwrite the DPR with the next AFB:

```
"BEGO"      ;ASCII header
PMRDSC      ;start the timer and A/D scanning (subroutine $019A)
$00000000   ;start channel=0
$0000000F   ;final channel=15
$00000001   ;one scan per buffer fill (>32 bytes cause buf error bit)
"ENDF"      ;ASCII terminator, timer stops here
"CONT"      ;loop continuously on AFB 0 ("SWEN" not needed to switch)
"ENDC"      ;ASCII terminator
```

Use the interrupt protocol as discussed before. Poll the data ready bits slowly or wait for a VMEbus interrupt from the 601. After each buffer fill, reset the appropriate ready bit, then transfer that buffer with moderate duty cycle to host memory while the alternate buffer fills. After all scans are taken, load and execute the following function command:

```
"RSET"      ;function command to stop scanning
"ENDC"      ;ASCII terminator
```

5.13 MC68901 Peripheral Controller Local Addresses

The information below is intended for users with local programs which need access to the MC68901 peripheral controller. The addresses below are hardware registers relative to the local 68010 CPU memory map. Normal squarewave output from any of the timers for A/D triggering is the delay mode. A timer control byte of $01 gives a divide-by-4 prescale. An additional divide-by-2 is the inherent in the output toggle, giving divide-by-8 as the smallest divisor of the 2.4576 MHz 68901 timer. Therefore the smallest period is about 3.255 microseconds using a $01 data byte. Consult the Motorola MC68901 Multi-Function Peripheral Advance Information, June 1988, publication ADI- 984R1 (order number MC68901/D). Byte access should be used or word access after adding one. All addresses are in hexadecimal.

Address	Description
00120001H	General Purpose I/O Register
00120003H	Active Edge Register
00120005H	Data Direction Register
00120007H	Interrupt Enable Register A
00120009H	Interrupt Enable Register B
0012000BH	Interrupt Pending Register A
0012000DH	Interrupt Pending Register B
0012000FH	Interrupt In-Service Register A
00120011H	Interrupt In-Service Register B
00120013H	Interrupt Mask Register A
00120015H	Interrupt Mask Register B
00120017H	Interrupt Vector Register
00120019H	Timer A Control Register
0012001BH	Timer B Control Register
0012001DH	Timers' C & D Control Register
0012001FH	Timer A Data Register
00120021H	Timer B Data Register
00120023H	Timer C Data Register
00120025H	Timer D Data Register
00120027H	Synchronous Character Register
00120029H	USART Control Register
0012002BH	Receiver Status Register
0012002DH	Transmitter Status Register
0012002FH	USART Data Register

5.14 Local RAM Reserved Locations

Local RAM from $20000 to $21000 is reserved for system variables. Although the full area is not completely used, future firmware revisions are likely to use more RAM. User written programs may need access to this area however system variables should not be changed without understanding the effects. User programs should be loaded at $21000 or above. A generous amount of local stack is available. The addresses below are in hexadecimal.

Address	Description
00020000H | 0002013FH	Reserved for Interrupt Vectors
00020140H | 000204CDH	Reserved for Local Variables
000204CEH | 00020FEFH	Reserved for Supervisor Stack
00021000H | | 0002FFFFH	Free RAM Area (for Program Development)

5.15 Interrupt Vector Assignments

These vector addresses are loaded by the EPROM after power up.

Address	Description
00020000H	Reset Exception Vector
00020004H	Reset Exception Vector
00020008H	Spurious Interrupt Vector
00020023H	
00020024H	Trace Exception Vector
00020028H	Spurious Interrupt Vector
0002007FH	
00020080H	TRAP #0 (Goto MONITOR) Exception Vector
00020084H	TRAP #1 Spurious Interrupt Vector
00020088H	TRAP #2 Spurious Interrupt Vector
0002008CH	TRAP #3 Spurious Interrupt Vector
00020090H	TRAP #4 Spurious Interrupt Vector
00020094H	TRAP #5 (Start XMTR) Exception Vector
00020098H	TRAP #6 Not defined
0002009CH	TRAP #7 Not defined
000200A0H	TRAP #8 Not defined
000200A4H	TRAP #9 Not defined
000200A8H	TRAP #10 Not defined
000200ACH	TRAP #11 Not defined
000200A0H	TRAP #12 Not defined
000200A4H	TRAP #13 Not defined
000200A8H	TRAP #14 Not defined
000200ACH	TRAP #15 Not defined
000200B0H	Reserved
00200FFH	
00020100H	GP I/O 0 Exception Vector
00020104H	GP I/O 1 Exception Vector
00020108H	GP I/O 2 Exception Vector
0002010CH	GP I/O 3 Exception Vector
00020110H	TIMER D Exception Vector
00020114H	TIMER C Exception Vector
00020118H	GP I/O 4 Exception Vector
0002011CH	VME HOST to 601 Exception Vector
00020120H	TIMER B Exception Vector
00020124H	TRANSMIT ERROR Exception Vector
00020128H	TRANSMIT BUFFER EMPTY Exception Vector
0002012CH	RECEIVE ERROR Exception Vector
00020130H	RECEIVE BUFFER FULL Exception Vector
00020134H	TIMER A Exception Vector
00020138H	END OF CONVERSION Exception Vector
0002013CH	END OF SCAN Exception Vector
00020140H	Reserved
0002015FH	

5.16 Stack Definition When at the Monitor Level

User programs may read the contents of the stack but should not modify them. Hexadecimal addresses are given.

Stack Pointer Address Offset	Content Description
0000H	SP - Stack Pointer when entered at MONITOR Level
0004H	VBR - Vector Base Register
0008H	D0 - Data Register #0
000CH	D1 - Data Register #1
0010H	D2 - Data Register #2
0014H	D3 - Data Register #3
0018H	D4 - Data Register #4
001CH	D5 - Data Register #5
0020H	D6 - Data Register #6
0024H	D7 - Data Register #7
0028H	A0 - Address Register #0
002CH	A1 - Address Register #1
0030H	A2 - Address Register #2
0034H	A3 - Address Register #3
0038H	A4 - Address Register #4
003CH	A5 - Address Register #5
0040H	A6 - Address Register #6
0044H	SR - Status Register when entered at MONITOR Level
0046H	PC - Program Counter when entered at MONITOR Level
004AH	FV - Format/Vector Offset Register
004CH	Stack Pointer after exiting MONITOR Level to EXECUTIVE Level - Top of Stack on Power Up -

5.17 Local CPU Wait States

Memory transfers using the local 68010 CPU have the following wait states:

EPROM or local RAM...0 wait states
Dual Port RAM...4 wait states at 125 nanoseconds per wait state due to arbitration logic.

LOCAL PROGRAMMING

6.0 Introduction

If you have decided to do local programming, your code must be stored either in the on-board EPROM or in the host when power is off. Host program storage is usually on disk and the program must be downloaded through the DPR into local RAM each time power is started. You may use the DPDL or XFRxxx subroutines for downloads. You must also decide if you wish to write your program from scratch (perhaps using the Monitor to debug it) or added on to the end of the EPROM binary code.

There are two strategies regarding your local code and A/D control. You may either write your own A/D subroutines or call those in the Executive EPROM, if they perform the task you need. If you call the Executive, parameter passing must be identical. If you write your own, the Executive EPROM source files may be used as an example. The source files are available on IBM-PC 5.25" disks to customers on request. Contact the home office Subsystems Application Engineering Department.

Either task requires an experienced programmer. If you are considering reprogramming the EPROM versus downloading, these are some alternatives:

1. The EPROM retains your program permanently and it is more immune to CPU lockups caused by transients. Therefore, EPROM code storage may be more reliable than host disk storage. The DPR download method (not using the local EPROM) requires a short loader program each time power is started.

2. Execution can be programmed to start automatically from EPROM, without loading any DPR commands.

3. The DPR code download method allows you to operate several programs in local memory one after the other in overlay mode. For example, a number of long, high precision linearization tables may be used for different sensor families. Using a full table (no piecewise linear interpolation) offers very high speed and accuracy. But it takes up much memory and may only allow room for one sensor type. Or you may use the larger 128 Kb EPROM to store several tables.

4. Once the S2 records are placed in DPR, the download is quick. The S2 ASCII records contain destination addresses and checksums for verification as they are converted to binary.

5. Execution never begins after a download unless that is programmed as the next function command. Therefore the host may take action in case the download gives a checksum error or a higher-priority task interrupts.

6.1 Local Subroutines

The following subroutines are listed in the EPROM source files and may be called from programs which the user has written and loaded into local memory. These subroutines should not be called from the DPR as Executive subroutines because their parameter-passing method is different than the DPR sequential list parameter syntax. Use the MAP file to determine the EPROM absolute address

These subroutines are generally divided into math routines and serial port I/O routines. The primary Monitor serial port uses two transmit and receive interrupt-driven ring buffers to avoid loss of characters during other CPU processing. This offers reliable response to the control-X hard reset. Most of these routines have been optimized for use under the C language or from assembly. It is possible to write local code entirely in C if the output is absolute image (ROMable) code. For some C compilers, you will need to rebuild the START.C module.

6.1.0 Serial Port Local Subroutines

Subroutine Name	Description
PR_MES	Sends an ASCII string to the serial port. The message must be terminated with $FF.
CRLF	Sends a carriage return ($0D) and line feed ($0A) to the serial port.
SPACE	Sends an ASCII space ($20) to the serial port.
ASCHEX	Converts a string of up to 8 ASCII characters to hexadecimal.
OUTCHR	Output a single character to the serial port.
PRNT8HX	Sends 8 ASCII hex characters to the serial port from a 32-bit binary number.
PRNT6HX	Same as above for 24-bit binary.
PRNT4HX	Same as above for 16-bit binary.
PRNT2HX	Same as above for 8-bit binary.
PRNT1HX	Same as above for 4-bit binary.
PROMPT	Sends a prompt message to the serial port and waits for serial input terminated by a carriage return.
INPLNE	Accepts a serial port input string terminated by carriage return to a user-defined buffer.
PMTCHR	Sends a string message to the serial port and waits for a single character input.
INPCHR	Waits for a single character input from the serial port.
RDATA	Waits for a single character from the auxiliary serial port at 4800 baud. This routine also invokes the software UART timing.
POLCHR	Try to input a single character from from the serial port input ring buffer. This routine does not wait for the character to be input. Useful for periodically polling a user during a continuous process.
ODATA	Sends a single character from the auxiliary serial port at 4800 baud. This routine invokes the software UART timing.

6.1.1 Arithmetic Local Subroutines

The following describes the EPROM floating point math library. You will need the Whitesmiths 68000 packages to fully understand these routines. Or you may code your own math library. The 601 does not use these routines as part of the Executive. Whitesmiths' address is:

Whitesmiths, Ltd., 59 Power Road, Westford, MA 01886 USA, Phone: (508) 692-7800

Subroutine Name	Description
ASCDBL	Converts a decimal ASCII string terminated by $00 into a double precision internal format.
DBLEXP	Converts an internal double precision value into a decimal ASCII string in exponential format.
DBLFIX	Converts an internal double precision value into a fixed point decimal ASCII string.
STRLEN	Scans an ASCII string and returns the number of characters before the NULL terminator.
CLPOLY	Computes an Nth order polynomial from a double precision coefficient table.
CALEXP	Computes a double precision exponential.
CALSQR	Double precision square.
CLSQRT	Square root.
CAL_LN	Natural log.
CALLOG	Log to the base ten.
CALSIN	Sine in radians.
CALCOS	Cosine in radians.
CALTAN	Tangent in radians.
CALCOT	Cotangent in radians.
CALSEC	Secant in radians.
CALCSC	Cosecant in radians.
CLSINH	Hyperbolic sine in radians.
CLCOSH	Hyperbolic cosine in radians.
CLTANH	Hyperbolic tangent in radians.

CALARC	Arc tangent in radians.
FLTADD	Floating point addition.
FLTSUB	Floating point subtraction.
FLTMUL	Floating point multiplication.
FLTDIV	Floating point division.
TRNDBL	Converts a double precision number to the nearest integer closer to zero.
RNDDBL	Increases the absolute value by 0.5 and converts to the integer nearest zero.

Local Programming Details

The following section gives more insight into local user programming, if it is required. The Executive DPR functions are all defined by EPROM firmware and may be completely redefined by user programs. The only items which cannot change are the three DPR hardware registers and the local registers.

Rules for user programs downloaded through the DPR:

User subroutines which are loaded as S2 ASCII records at the DPR BASE + 0 and then are downloaded into local RAM using the "DPDL", "ENDC" command should obey the following rules:

1. Your code must be a SUBROUTINE and may be called from the DPR simply by putting its longword local entry address in an Application Function Block as follows:

```
"BEG0"      ;longwords decrementing from BASE+$FFEC
$00028000   ;your choice of local entry JSR address
"ENDF"      ;marks end of AFB
"STRS"      ;calls the code at $28000
"ENDC"      ;marks end of function command
```

Host resets ACK bit in HSTSTA (BASE+$FFF6,7)
Host interrupt to 601 written at BASE+$FFFC,D
Host polls for ACK bit set. Code should be running.
Check HSTSTA/ADSTAT error bits then reset ACK.

2. This discussion here applies only to user-downloaded code called from the DPR, not from the Monitor. Although your code must be called from the DPR, it may be loaded from either the DPR (DPDL or XFRxx commands) or from the Monitor (SPDL command). Your routine should be a subroutine with an RTS return (or conditional return) after your code executes. Other subroutines may of course be nested within your code.

3. Before your routine exits, do not permanently alter the stack contents, stack pointer or local system control variables in local RAM below location $21000. Local 68901 interrupts may be changed as long as the existing Executive interrupt system is not altered. Otherwise the host may lose control.

4. Remember to use only one control character (either CR or LF) between S2 records. The address information contained in the S2 records determines where the code loads in local memory. These addresses are formed by linker commands or ORG statements. You must decide where in local RAM to load your code

based on whether A/D scans use the local A/D buffer first. If A/D scans are sent to the local buffer first, your code should be ORG'd to load properly in high local RAM away from A/D data which builds upwards from location $21000.

Calculate the amount of RAM required for local A/D data buffering based on the number of channels times the number of scans using 2 bytes per A/D sample. If local A/D buffering is not used (A/D scans are sent only to the DPR buffers), your code may load starting at location $21000. You may also change the A/D local buffer pointer using the INBFPT DPR subroutine.

5. Avoid entry-level external function parameters passed on the stack as used in C programs until you understand them completely. Instead, write your code so that external parameters (if needed) may be passed through low DPR. (The parameters may overwrite the old S2 download records after the download but before you call your local code).

All CPU registers may be used in your program and they do not have to be saved first. Locations BASE + $FFF0-$FFF3 may be programmed as a user-designed control/status area to alert a polling host program. Or interrupts may be sent to the host from your local program.

Local high-level language programs

It is possible to write your local programs as C functions or as other high level language subroutines. Parameters may optionally be passed on the stack and in fact, most local EPROM subroutines were designed with this in mind. If your code is written in C, DPR locations and local registers would be controlled by pointers. If you decide to use C with sequential DPR function parameters, register A6 is used as a frame pointer which must be linked then unlinked with the proper number of longword DPR parameters (consult your compiler documentation). Compiled C programs downloaded through the DPR or extended onto the EPROM should have the following form:

```
/* Don't use main() !! */

    func() /* no parameters passed until you know how */

    {
       int a,b;          /* local variables as needed */
       char c,d;

       statement 1;
            :
       statement n;
    }              /* unlink then RTS */
```

If your C program starts with variable declarations (which reserve stack space), examine the assembly output to determine the actual entry point. Use linker commands to get the S2 records to produce the proper load address. Examine the list and MAP files to determine the address of the entry label. Your C program should NOT include a main() function since this may improperly allocate system linkage. Calls to EPROM subroutines must obey the usual external declarations and label syntax.

6. If necessary, your local program may jump to the Rev.1 EPROM location $18 (or $1C, revision 2.x EPROM) after execution. This is an abort entry location to be used with possible error recovery code. JMP $18 (or $1C, rev 2.x) is the ABREXC label in the source code. If the Executive is ON after a jump to ABREXC, the stack will be restored and the 601 will stop at the Exec command level. If the Exec is OFF, a soft reset to the Monitor will occur. The host will then have to do an EXEC ON<cr> cycle to recover control.

Rules for programs executed from the Monitor:

1. Code may be executed using the GO <address> Monitor command. First turn the Executive ON then control-C back to the Monitor. If you forget to turn on the EXEC, GO will respond with a question mark (?).

2. GO <adrs> code should NOT be a subroutine although subroutines may be nested within your code. Instead, the code should terminate with a JMP ABREXC instruction. When this occurs, the 601 may be in the Executive and can be transitioned back to the Monitor with control-C.

3. The "EXC>" prompt will not be displayed after the JMP ABREXC instruction if the Executive is entered. Unless you have programmed other serial port activity, it will not be obvious when your code finishes. If you wish to be notified just before this exit jump, send a message out the serial port using multiple calls to the OUTCHR subroutine at $000C4. When called, OUTCHR sends a single character contained in register D0. See the EPROM code.

4. The Monitor control-C function will still be available while your code executes. By examining the PC register, you can determine where your code is running by comparing the PC to your listing. Trace and breakpoint execution also are available.

Saving CPU Registers

No CPU registers have to be saved by user written subroutines. Although all subroutines provided in the DVME-601 EPROM show all registers being saved, this is not a requirement. If the correct value of the stack pointer is in question, the user can safely re-enter the EXEC by jumping to location ABREXC ($001Ch) when exiting his code.

Parameter Passing

Users may write local subroutines which pass longword input parameters after the longword subroutine address in an AFB. The subroutine address must be on an even boundary and is relative to the local CPU. This discussion assumes that when defining the Application Function Block, the correct arguments were passed through the Dual Port RAM to the DVME-601 after the subroutine entry address. Argument values are passed to subroutines utilizing the Function Block pointer register A4.

The EXEC calls subroutines which are in the Function Block(s) as follows. The EXEC uses address register A4 to point to its current position in the active Function Block. Longword MOVE.L's with auto increment are used to access the information from the internal copy of the Function Block after copying from the DPR. In the EPROM EXEC module, you will find the following code:

```
MOVE.L    (A4)+,A5    * Get pointer to next subroutine
MOVE.L    A4,-(SP)    * Reg A4 now points to next subr or
*                       to first argument of the subr.
JSR       (A5)        * Call the Function Block subroutine

MOVE.L    (SP)+,A4    * Restore the original (or updated)
*                       Function Block pointer
```

Note that the Function Block pointer is saved on the stack. Given this sequence of instructions, the user can pass arguments to his subroutine via the Function Block pointer, register A4. An example can best describe how this can be accomplished:

```
USER_SUBR:
                LINK    A6,#-4          * Reserve space on the stack
                MOVE.L  D0,(SP)         * for register(s) - save them

                MOVE.L  (A4)+,D0        * Get first argument
```

*STRCHN is the start channel address register

```
                MOVE.L  D0,STRCHN * NOTE Reg A4 has been updated
                MOVE.L  (A4)+,D0  * Get second argument
```

*FINCHN is the final channel address register

```
                MOVE.L  D0,FINCHN * NOTE Reg A4 has been updated

                MOVE.L  A4,8(A6)  * Update the saved version of
*                                   Reg A4. This MUST be done.

                MOVE.L  (SP),D0   * Restore D0
                UNLK    A6        * Restore stack pointer
                RTS
```

It is important to recognize that in the subroutine, register A4 was auto-incremented by FOUR (.L) each time an argument was passed. This insured that upon return to the EXEC, Register A4 was pointing to the next subroutine in the Function Block. In order for the EXEC to see this change in register A4, the save (on the stack) version of register A4 must be updated using the instruction - MOVE.L A4,8(A6).

The sequential list method of passing input parameters was designed for high level languages which use A6 as a frame pointer. The parameter-passing for C functions is taken care of automatically as long as the number of parameters in the sequential list matches the number of parameters input to the function. Local declarations in the function must accomodate 32-bit longwords. Normally these would be long int declarations however check your C compiler's assembly output to be sure. If you cannot easily declare 32-bit parameters, pass the parameters elsewhere in the DPR using pointers and make your function have zero parameters. That is, the function definition should be: func() with no return values passed through the stack. Values may be returned through the DPR with status bits or a VMEbus interrupt to alert the host.

EPROM Checksum calculation

The DVME-601 calculates the Checksum by performing a 16 bit addition - one byte at a time. PROM locations 0 through $FFFD are summed. This 16 bit sum is then compared to the 16 bits found at PROM locations $FFFE - $FFFF. Since there are two PROMs on the DVME-601, be sure the MSB and LSB parts of the Checksum are loaded into the proper PROM. The PROM marked PPE-13171-1 holds the MSB and PPE-13171-2 holds the LSB of the Checksum.

Note - The checksum computation and load is performed manually on the EPROM programmer. You must have a programmer which includes manual byte edit capability. If that is not available, the checksum may be added into the assembly source after making the first checksum computation of the absolute executable binary image file. Or you may simply remove the checksum operation in the source code. The 601 continues with normal operation if there is a checksum error.

Local DPR Downloaded Code

	Written in Assembly	Written in "C"
No in-line DPR Params	All Registers free except A7=stack pointer	A6=frame pointer
In-line DPR params	A6=frame pointer (suggested, see example) A4=AFB pointer A7=stack pointer	A6=frame pointer. Must use assembly module to call (A4). Make sure compiler doesn't crash A4 as register declaration.

6.1.2 Breakpoint Timing

When analyzing local user-written assembly code to run the A/D converter, breakpoint operation requires additional time between user instructions to test for the breakpoint. Therefore input A/D bandwidth will suffer. Breakpoint should only be used to verify proper code operation. A very rough guideline is 75 microseconds between instructions under breakpoint.

When a breakpoint address is set, the local 68010 CPU runs with its trace bit on. The CPU will do a software exception trap with each instruction. The Monitor code then will do a compare of the instruction address with the breakpoint address. If there is no match, the Monitor allows execution of the next instruction. If there is an address match, the code breaks to the Monitor, where the program can be analyzed. (You will need your program listing file.) Execution resumes with the G<cr> command alone with no address. Use breakpoint to verify your code then always run high speed A/D data without breakpoint. Use the NOBR Monitor command.

6.1.3 VMEbus Interrupt Operation

The following items may be reviewed for users who want an interrupt driven software interface to the 601. Interrupts are optional and should only be considered if host system performance demands interrupts. These are VMEbus interrupts sent from the 601. They should not be confused with local 601 interrupts used by the on-board 68010. Note that interrupts to a Real Time Operating System can be surprisingly slow. consider VMEbus interrupts to enhance total system synchornization and avoid lost A/D data.

Before you try VMEbus interrupts, always get your application to run flawlwssly in status polled mode without VMEbus interrupts. Use the HSTSTA status bits to detect when an AFB is defined, started, active, and complete. The "active" flags, HSTSTA 14 and 15, indicate which AFB is being executed or will be executed next. For two defined AFB's, these flags switch as an AFB is started. The default for these flags is to indicate the last AFB defined (the lowest in the DPR memory).

The HSTSTA "done" bits 6 and 7 indicate that the end of the AFB has been reached. For an AFB which is executed only once, this is similar to a "A>" DOS prompt, indicating that the 601 Executive is finished running the AFB and is back at the command level.

Remember that the 601 does not reset most status bits since it has no way of knowing when your host program reads them. Reset them after your read them if your want to detect them the next time the 601 Exec sets them. The only item that sends VMEbus interrupts are the ADSTAT bits 1 and 2. A local user program could also send interrupts.

Quick checklist for DVME-601 to VMEbus interrupts:

1. Is the desired IRQ line and jumper properly connected?

2. Is bit 7 of BASE + $FFF8,9 set to 1? This gates out the IRQ when the local interrupt register ($EXXXX) is written to. The A/D routines write to this register whenever either of the data ready bits (ADSTAT bits 1 or 2) are set to 1. Optional local user-written code may also send VMEbus interrupts but without setting the A/D ready bits.

3. Is the CPU board which will handle the 601 interrupt, also the current bus master? Interrupts sent to the wrong CPU board master will have undefined results. Remember that the IRQ lines are read by all CPU's.

4. Is the CPU board interrupt mask in the status register less than the IRQ level? Too high a mask will ignore interrupts.

5. Is the proper vector ID loaded at BASE + $FFFA,B on the 601? This should be the ISR longword address (or VBR longword offset) divided by 4 (2 right shifts with zero MSB fill). Note that the Motorola vector ID's are given in decimal. For ID 64, load $40 hex meaning ISR address offset $100 hex. The lower 8 bits of the ID code can be loaded and read back using word addressing from your Debugger to confirm that the register is there. Don't write a code greater than $00FF.

If you get spurious interrupts, the hardware is probably working but the ID or ISR or Real Time OS (RTOS) setup and allocation is probably faulty. RTOS usage can be very demanding. Always start with fully documented known working device driver source code.

6. Is the 601 really sending VMEbus interrupts? Confirm with an oscilloscope on the interrupt lines. Get your 601 host program to work properly without interrupts first under polled status mode. To confirm repetitive A/D writes to the DPR, the DPR may be zeroed out (block fill) between scans. The EOC output (pin J3-7) may be watched with an oscilloscope to confirm A/D activity.

7. Is the host CPU Vector Base Register loaded properly? Is the user-written host interrupt service routine loaded in memory at the absolute address pointed to by the host CPU VBR plus the vector ID code times 4? If necessary, compare your ISR lost file with the disassembly from your host debegger. In the case of a Real Time Operating System, verify priorities, task initialization, memory allocation, etc., etc.

What exactly does your ISR code do? At the very least, it should signal the OS (perhaps with a message structure, flag or mailbox) that the 601 has DPR data ready and needs service. If there's enough time available in the ISR, go ahead and transfer the DPR data. Use moderate duty cycle if the 601 is simultaneously trying to fill the alternate buffer. Otherwise a MOVE.W (A0)+, (A1)+ type block transfer burst is acceptable. Make sure no longword transfers are used on the DPR. Remember to reset the appropriate data ready bit if wait-for-reset mode is used. If there's enough time, test and reset the other DPR status bits to detect errors. The 601 will continue even if you don't reset them.

Does your ISR disable other interrupts then reenable them? Does your ISR finish with RTE? Are register properly LINKed and UNLKed? Is the interrupt priority mask properly managed and restored during the ISR? What are the consequences of another interrupt received (including an A/D interrupt) while processing the 601? Is your ISR fully reentrant with registers properly saved and restored?

8. The 601 asserts an 8-bit vector ID code only on data lines D0-D7. This vector ID is stored in U35. Some recent implementations of the VMEbus interrupt may look for 16 or 32 bit interrupt codes. Most CPU boards should be able to accomodate the 8-bit ID code. Remember that the 601 is a slave and cannot control the bus. It needs an external CPU board to transfer the data.

9. Are the IACK, IACKIN and IACKOUT lines daisy-chained on board slots between the 601 and the CPU board? For empty slots, make sure these lines are jumpered on the backplane.

6.2 Local A/D Data Buffer Control

A/D scans commanded from the DPR Executive may send the A/D data to local (private) RAM or to the DPR. Scans sent to private RAM use a local buffer structure which is managed by the Executive. Local programs written by the user may need to reassign the default buffer address used by the Executive if Executive routines will be called by the local program. The addresses below contain parameters which may be modified before calling an Executive routine and returning control to a local program. This allows the local program to place the buffer so as not to conflict with local memory usage.

locptr equ $0021E

This EPROM address points to the location which holds the pointer to the local data buffer area. (A "pointer to a pointer"). This address never changes but the contents of the pointer may be changed by your local program. Always use the indirect pointer format since later software revisions may change the contents of the address pointer loaded into RAM.

cntptr equ $00222

This address points to the location which holds the current data "object" count (words, longwords or doubles) for the local buffer. The data type is set by the buffer definition Executive commands.

6.3 Thermocouple Temperature Measurement

Linearizing thermocouples is a common arithmetic function after A/D scanning and before transfer to the host. The discussion below covers many items which must be considered connecting thermocouples through the DVME-643 slave MUX board using either a DVME-601 or 611/612 master A/D board. Other sensor types will use similar techniques.

DATEL's DVME-643T low level slave multiplexer board directly connects to thermocouples for temperature measurement via detachable screw terminals. The DVME-643 may be used as an input source for either the DATEL DVME-611/612 or DVME-601 A/D boards. Any of these A/D boards operate the 643 in the same manner. The 643 is connected to the host A/D board via the channel expansion bus which fully controls channel selection and analog multiplexing. The expansion bus uses flat cable front panel connectors and may be extended up to 10 slave MUX boards within two meters away from the A/D board. The only connection the DVME-643 makes to the VMEbus is for +5V dc power for its on-board DC/DC power converter.

The DVME-611 and 612 A/D boards require thermocouple math to be done in the host whereas the 601 may optionally do the math using its on-board CPU before transfer to the host. Since the 601 also includes 5 programmable TTL discrete outputs, fully independent limit testing, host interrupt alarm and external device operation may be performed for process control applications.

6.3.1 Input Channels

The DVME-643T includes eight expansion input channels with channel-to-channel galvanic isolation, and gain programmable input amplifiers. These functions are provided in DATEL SCM-100 series encapsulated Signal Conditioning Modules. Each amplifier may be jumpered for a gain of X50 or X100 and may be individually calibrated for offset and gain trim. These adjustments are adjacent to each SCM on the board and are found using the schematic and assembly drawings for the DVME-643T. The full scale output

range from each SCM is ±5 Volts maximum. Isolated dc power is also available from each channel for optional sensor excitation (not used for thermocouples).

The 643 also includes a temperature sensor mounted adjacent to the screw terminals used for thermocouple input channels. This sensor provides electronic cold junction compensation (CJC) and has a raw output of 20.48 milliVolts per degree Celsius referred to zero degrees C. This may be expressed as follows:

$$\text{CJC output(in mV)} = (\text{Sensor temp.} - 0\ \text{deg.C}) \times 20.48\ \text{mV}$$

or

$$\text{CJC Temp. (in deg.C)} = \text{CJC amplifier output (mV)}/20.48\ \text{mV}$$

Thus if the board is operated at +20 °C (near room temperature), the CJC output presented to the PGA input will be +0.4096 Volts dc. The CJC temperature range is compatible to that of the host DVME-643T which is 0 to +60 °C. At +60 °C, the maximum CJC output will be about 1.2 volts dc. The CJC will remain accurate with negative readings a few degrees below 0 °C. The CJC circuit may be calibrated for gain. Once calibrated, the CJC circuit is accurate to better than ±0.5 °C.

There is one CJC sensor for each DVME-643 board. CJC correction should be done for each board since slight temperature gradients may exist between rows of screw terminals in adjacent boards, especially if there is poor cooling circulation. DVME-643's not sharing the same VME card cage should definitely sample each CJC sensor. The CJC readings should be compared to detect if any CJC channel fails. Normal readings within a few degrees should be expected. The CJC may be sampled once with all eight thermocouple input channels or less often in known stable environments. Board temperature may also be used to detect overheating of the host computer.

6.3.2 Cold Junction Compensation

Thermocouples can only indicate the absolute temperature difference between the input thermocouple and the reference TC. Since the "reference" junction is really the TC-to-copper junctions formed at the screw terminals, these junctions must have a known temperature. The effect of increasing the "reference" junction temperature above 0 °C is to reduce the net millivolt output from the three series-connected junctions (i.e., the input TC and the two screw terminal junctions). This reference junction cannot be directly measured. Instead it must be derived by an inverse temperature-to-millivolt function. To adjust the millivolts output to what would normally occur if the reference were at zero degrees Celsius (where most thermocouples are calibrated), the temperature in equivalent millivolts for the type of input junction used must be ADDED to the net output millivolts. Thus the CJC-compensated millivolt output of the input TC is as follows:

$$\text{CJC'ed output (in mV)} = \text{Raw input (mV)} + \text{CJC sensor temp. (in mV)}$$

Notice that the correction is all in millivolts dc, not in temperature. This reflects the actual series-connected input circuit and is necessary because of the mV-to-temp. nonlinearity.

6.3.3 Channel Addressing

Address codes distributed along the channel expansion bus from the host A/D board select both the thermocouple input channels and the CJC sensor output on each DVME-643T. Placing an address on the expansion bus multiplexes the corresonding channel outputs onto a differential analog signal bus which is received by the PGA on the A/D board. Channel addressing registers on the A/D board may be initialized from the host and can automatically sequence with each A/D sample. Channel addresses are decoded on each MUX board with a Base Channel Address selected by decoder switches. The actual addresses used are a positive offset from the selected Base Channel Address (BASE + local channel address).

Channel addresses Base + 0 through Base + 7 are used for the eight thermocouples and addresses Base + 8 through Base + 15 are for the single CJC channel. Since the top bit of the channel address (bit 3) is used to switch the CJC output, any address from 8 to 15 will select the CJC.

For multiple DVME-643T's sharing the same channel expansion bus, the channel address map will repeat every 16 channels with 8 thermocouple channels followed by 8 CJC channels for that board. The first 8, 16, or 32 channels in the address map are always assigned to the master A/D board, depending on the board type.

The resulting map will appear as follows:

Channels 00 - 15: A/D Master board channels (depends on board)
Channels 16 - 23: DVME-643T #1 channels 0 - 7
Channels 24 - 31: DVME-643T #1 CJC
Channels 32 - 39: DVME-643T #2 channels 0 - 7
Channels 40 - 47: DVME-643T #2 CJC

Other MUX boards, such as the DVME-641 and 645 may be mixed on the same expansion bus if the addressing, gains and full scale ranges are compatible. Users setting up the channel base address for each DVME-643 slave multiplexer must be sure the addresses do not overlap. This will prevent two boards from connecting their outputs onto the bus simultaneously.

Notice that the DVME-643 addressing must start on an address boundary of 16 because the bottom four bits (bit 3 thru bit 0) are not decoded by the base address switches. For example, one DVME-643 cannot be addressed for channels 24 - 39 since 24 is a non-decodable base address.

Other DVME-64X series MUX slaves have base address decoders with different address resolution and must start on boundaries compatible for their decoders. The user should construct a full analog expansion channel address map including the host A/D board's channels to enhance compatibility and avoid address "holes" in the map.

6.3.4 Gain and Input Ranges

Raw thermocouple inputs first pass through the SCM-series isolation preamplifiers before being multiplexed onto the channel expansion bus. Thus thermocouple inputs will be subject to a gain of X50 or X100 before they reach the A/D input. The CJC output does not pass through these preamps but does pass through the A/D board's PGA.

Both the preamplified output from the SCM's and the CJC output also pass through whatever Programmable Gain Amplifier is used on the A/D board. Therefore the PGA gain must be considered for both the CJC and the amplified TC channel. For the DVME-611/612, the software PGA will offer gains up to X128. The DVME-601 PGA may be resistor-selected for gains from X1 to X1000. In addition, the amplified signal must use the input range jumpered for that A/D type.

Thus there are three items (SCM gain, PGA gain and A/D bit weighting) which must be considered when converting from thermocouple millivolts to A/D binary output. The A/D bit weighting of course is a function of the full scale A/D input range and the number of code states. Because of this flexibility, it is possible to adjust the system for higher resolution in a narrow part of the input temperature range.

6.3.5 Thermocouple Arithmetic Operations

Two arithmetic functions, cold junction compensation and linearization, must be performed to obtain temperature outputs. Celsius or Fahrenheit conversion may also be a part of these math operations. CJC may be done externally using an actual ice-point reference or it can use the DVME-643T on-board CJC sensor. The latter method will all be done in software.

In the following description, both forward and inverse temperature to millivolt conversions are referenced. They may be performed either by two polynomials or by two tables. A set of tables is handy for determining appropriate gains and input ranges. The National Bureau of Standards (Boulder, CO) Monograph 125 may be used and includes both tables and polynomial information.

After all channels and the CJC are calibrated, the procedure using the on-board CJC sensor is as follows:

1. Select the CJC channel address (Base + 8).

2. Make an A/D conversion to determine the ambient temperature adjacent to the screw terminals.

3. Convert the binary CJC value to temperature using the following formula:

$$\text{CJC Temp. (deg.C)} = \frac{(\text{No. of A/D counts} \times \text{A/D bit weight in mV})}{\text{PGA gain} \times 20.48}$$

where,

$$\text{A/D bit weight (mV)} = \frac{\text{Full scale input range in mV}}{(2 \text{ ^ no. bits of resolution})}$$

where the "^" means exponentiation.

For example if we have a ±5 Volts input 14-bit A/D converter, the full scale range span is 10 Volts (not 5V) and there are 2^14 = 16,384 code states. This gives 10,000 mV/16,384 or a bit weight of about 0.61035 mV per bit.

4. Convert the CJC temperature to the equivalent millivolts for the type of thermocouple chosen. This is an inverse mV-to-deg. C. conversion.

5. Now determine the millivolts from the thermocouple channel. Use the equations above except that the SCM gain (X50 or X100) must be used as a divisor and the 20.48 mV per deg. C CJC divisor is not used. Don't forget the PGA gain. Also, observe the A/D polarity mechanism (either 2's complement, offset binary or sign/magnitude) if it is bipolar for cold temperatures.

That is,

$$\text{Raw TC (in mV)} = \frac{\text{no. of A/D counts} \times \text{A/D bit weight (in mV)}}{\text{PGA gain} \times \text{SCM gain}}$$

Using the example above, assume the A/D bit weight is 0.61035 mV per bit. Now assume that the PGA gain was 2 and the SCM gain was 100. If the A/D shows a binary output of +10,000 counts, this works out to:

$$\text{Raw TC (mV)} = \frac{10{,}000 \times 0.61035\text{mV} = 30.6525\text{ mV}}{2 \times 100}$$

For a type J thermocouple, this is roughly 558 °C before the CJC correction. Notice that the effect of reducing the gains would lower the number of A/D counts which would increase the input temperature range. Higher gain offers higher resolution but may truncate the input temperature range.

6. Add the raw thermocouple millivolts to the CJC millivolts:

CJC'ed output (in mV) = Raw input (mV) + CJC sensor temp. (in mV)

This gives a result which has been corrected for the cold junction not being at 0 °C.

7. Convert this millivolt sum back to Celsius temperature to achieve a linearized, CJC'd result.

8. Convert the Celsius temperature to Fahrenheit if required. The conversions are:

$$C = (F - 32) \times (5/9) \text{ or } F = (1.8 \times C) + 32$$

APPLICATION DEVELOPMENT

7.0 External Triggering

External events may be used to start a single A/D conversion or a multichannel scan of N conversions. Common applications trigger A/D conversion when a limit switch is cycled, when a photoconductor is activated or from external digital logic. Before connection to the DVME-601 trigger input, any such external signal must first be converted to a negative-going pulse with proper logic levels and TTL risetime/falltime characteristics. The trigger input specifications are as follows:

Logic "1" level	+2.0 to +5.5 Volts
Logic "0" level	+0.8 to 0.0 Volts
Input impedance	4.7 Kilohm pullup resistor to +5V dc
Falltime	100 nanoseconds max.
Trigger width	100 nanoseconds min., 2 microseconds max.

Triggering occurs on the falling edge.

Note that making connections directly from a limit switch may produce switch bounce transients which the 601 will interpret as additional false triggers. The switch output must first be conditioned to TTL characteristics, probably with noise-rejecting Schmitt one-shot devices. Similarly, dc voltage sources must first be conditioned by a voltage comparator with hysteresis to produce a sufficiently fast TTL edge signal. Note that external logic ground must connect to the DVME-601 ground input.

7.1 Trigger Inputs

The trigger inputs are used both for external events or for the local on-board start timer output. External trigger inputs may be made at two input pins, J2-10 or J3-17. J2-10 must be capacitor-coupled (1000 pF, 50V ceramic) to J3-17 to produce a sufficiently short pulse from squarewave outputs such as the 68901 timer. This coupling capacitor may have already been added to your board - examine the components near J3 and the R11 pullup resistor. Both trigger pins use the front panel D connectors. Local timer outputs use only one of these pins, J2-10.

Any of the digital grounds on J2 or J3 may be used for the trigger ground. Use shielded cables to avoid radiating trigger noise into the nearby analog inputs.

If no channel expansion slave MUX is used, external trigger inputs may be connected to J3-17. If a slave MUX board is used, external triggers may be connected to either J2-10 on the DVME-601 or to the trigger input connector on any of the slave MUX boards. The slave MUX trigger input is usually on a 9-pin front panel D connector and it is transmitted through the J3 channel expansion bus.

All external slave MUX triggers are wired in parallel and act as an open-collector wire-OR'd connection using a common pullup resistor on the 601. Any trigger will start conversion as programmed on the host 601 and are not dependent on the channel boundaries of the board accepting the triggers. This allows triggers on the third MUX board to start a scan of channels on the 601 or vice versa.

If the local 68901 is used as a trigger, make sure no other logic device is pulling the trigger line low. You may want to disconnect the remote trigger line at the J3 connector to avoid false triggers or logic incompatibilities.

7.2 Trigger Programming

Two command modes controlled by the A/D command mode register enable external triggering. This register is local address $0AXXXX where XX hexadecimal address nibbles may contain any mix of 0's or 1's.

When bit 1 in this register equals 1, the trigger is enabled. Writing a one arms the trigger, and waits for the falling edge of the trigger to start the actual conversion. Command bit 2 must equal 1 to enable A/D conversions.

Trigger modes are fully controlled by Executive subroutines however they are explained here at the register level for better understanding. The two possible trigger command modes in location $0AXXXX are:

Single Conversion Trigger

Mode X--X10 starts one conversion from a single trigger. After the A/D data has been read, the trigger remains armed to accept more triggers. During conversion, new triggers are ignored. A zero in command bit 2 will disable the A/D converter and ignore triggers.

Scan Trigger

Mode X--X11 allows one trigger to start the first conversion of a multichannel scan until the EOS signal occurs. After the first trigger, local 601 logic automatically starts successive conversions by repeated reads of the A/D data register at location $10XXXX. The Executive subroutine (or the user's program) running this mode must store each new A/D data sample in a memory buffer. The routine must detect the EOS signal to return from the subroutine after the last conversion.

Notice: The user's external circuits must inhibit additional triggers after the first trigger and during the scan in mode X--X11. The trigger input remains live and more random triggers may cause timing difficulties.

7.3 Pacer Clock Triggering

The local timer in the 68901 Peripheral Controller may be used as a source of stable, crystal-controlled external triggers. The timer has three 8-bit sections which may be used singly or cascaded. Normally the last stage in the timer chain is timer A. It's output on J2-19 may be externally jumpered on the connector to the trigger input, J2-10. When using the timer as a trigger source, trigger input J3-17 is not used.

The timer is programmed for mode and period according to the 68901 programming instructions, reproduced here in the Appendix, courtesy of Motorola, Inc. Executive subroutines may be used to do this programming or it can be done from a user's local program. Note that 68901 programming must use byte access (.B) instructions or read-modify-write.

7.4 Trigger Rates

Sampling Theory

A/D conversion devices are sampled data systems. They must obey the Claude Shannon sampling theorem which simply states that a band-stopped signal may be fully recovered if the sample rate is at least twice its highest frequency spectra. If it is not sampled at a sufficiently high rate, periodic magnitude errors may occur caused by alias frequency folding. This requirement is also variously referred to as the Nyquist criteria. It is extremely difficult by analytical means to separate alias noise from true data in an output array.

As a practical matter, this sampling theorem does not merely say to sample at more than twice the frequency of interest. It says to be sure there are NO input frequency components present at more than one half the sample rate. For a wide bandwidth input signal with unpredictable spectral content, users must employ a special external hardware filter on each channel to sharply attenuate signals well below the upper frequency limit. Wideband signals of unexpected spectral content occur frequently in robotics, chromatography, image scanning, resonance analysis and materials testing.

Users should also be aware of anything in the input circuit which could generate wide-band signals from inputs with known spectral content. The simple act of allowing wideband signals to exceed the full scale input range of preamplifiers may generate high frequency spectra as the input stages go into clipping saturation. Channel switching a few multiplexed channels at a frequency which is harmonically related to input spectra may also produce alias noise.

Anti-alias filters often use multiple poles to generate steep rolloff rates below the input frequency limit. If the rolloff rate is too shallow (too few poles), the corner frequency must be started so early that the full input bandwidth is not available without passband magnitude attenuation. Such filters should not produce substantial output ringing from high rate step inputs or from input signal clipping.

Finally, the sampling theorem applies to the A/D converter itself and to each channel approximately divided by the number of channels. Since the DVME-601 is a switched sampling multichannel system, input bandwidth on any one channel must reflect the net sample rate at that channel, not the A/D itself. For example, an A/D which can sample and transfer to memory one sample every 25 microseconds will have roughly a 250 microsecond throughput (4 KHz sample rate) on any one channel in a 10-channel continuously-scanned system. In this example, the input frequency per channel cannot exceed 2 KHz. All spectral components beyond 2 KHz must be attenuated to zero on each input channel. Any signal beyond 2 KHz, even if only a few tens of millivolts, will produce low level alias noise in the data. This noise robs the output data of resolution.

Triggering for Spectral Applications

Users with critical high resolution spectral applications should consider using full scale two-tone intermodulation testing from high purity generators. This will establish a baseline of residual dynamic performance for your particular configuration. The DVME-601's generous local RAM and DPR memories are ideal for producing large data arrays for high resolution FFT analysis.

Users attempting to do medium frequency spectral analysis applications should use high stability trigger sources. Any phase noise jitter in the trigger will translate into spectral data errors. Random or unknown delays introduced by the user's local data transfer software may also create spectral noise. Oscilloscope observation of the A/D's EOC output on J3-7 is suggested if you wish to observe A/D software timing consistency.

7.5 Contiguous Channel Mapping for DVME-64X MUX Expanders

The A/D board channels always start at channel 0 and end according to their channel capacity and A/D board type. A typical master-slave map is as follows (addressing is in hex):

Unused MUX #2	Channel Address: $3F $30 (base address)
MUX #1	$2F $20 (base address)
"hole"	empty if A/D board address stops at $07
DVME-601	$07 $0

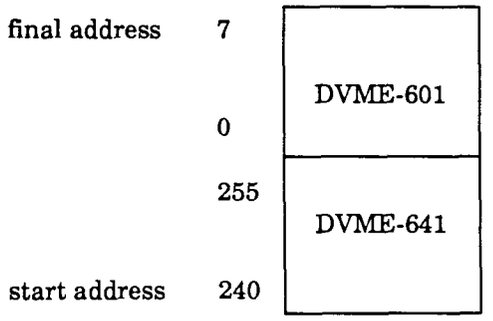
Where "MUX" means DVME-641, -643 or -645 slave expander boards.

The DVME-641 MUX channel addressing must start on a channel address which is evenly divisible by 16 (differential) or 32 (single-ended). If the 641 mapping follows the 601's, this will leave an address "hole" with no channels following the 601's local channels of 0-7.

Scanning through the holes where no channels appear can cause settling delay problems when a real channel is reacquired. The disconnected inputs in the holes may cause the A/D PGA input stages to saturate. The MUX's may be addressed first to prevent "holes" in the channel address map. If desired, MUX channels may be addressed high in the map so that they are contiguous to the A/D channels as the channel sequencer wraps around from $FF to $00, modulo 256. The MUX start channel is loaded at $F0 for example and the A/D board final channel at $07. The EOS comparator only tests for channel equality and doesn't care if current addresses are greater than the final channel.

For the DVME-601 local EPROM A/D subroutines however, some of the A/D subroutines do care that the start address is greater than the final address because they do a subtraction to determine how many samples to scan for. This has nothing to do with the hardware address counter logic; it's all in the subroutines (and therefore you could rewrite it). To use the standard subroutines, use a longword start channel address parameter that has been "minus sign extended" as the start channel. Example: If the MUX start channel address is $E0, use a longword address parameter of $FFFF FFE0 (not $0000 00E0) as the start channel address in the AFB subroutine call.

The following example illustrates this technique:



7.6 DVME-601 EXEC ON/OFF Timing from the DPR

When the EXEC ON and EXEC OFF commands are given from the DPR, extra time should be allowed before starting another command. The HSTSTA ACK bit 0 responds quickly after the command interrupt is written. But recall that ACK only means that the command has transferred successfully to internal RAM. ACK does not mean that the command is done or executed properly.

In particular EXEC OFF does a full software reset including memory test and reload of the stack pointer, VBR and 68901 interrupt registers. A VMEbus command interrupt to the 601 during this period could cause undefined operation and a host RESET instruction might be needed to recover control. Allow about 2 full seconds for complete recycle.

Similarly, EXEC ON takes more time than is indicated by the ACK bit. While a memory test is not done, about 10 milliseconds should be allowed before another command interrupt for internal Executive setup.

7.7 Logical DPR Addressing

The DVME-601 is a high speed memory-mapped peripheral similar to a disk drive board. It must be

mapped at an absolute physical address and requires both the VMEbus address and address modifier lines driven properly for successful operation. On some Operating Systems (notably UNIX), you must convert the physical address to a logical address using a conversion function supplied with your OS. These functions have names such as phys(addr), map(addr), mmap(), valloc(), etc. These functions would replace the normal "C" pointer operations. On some systems, this access is called "shared physical memory". On other systems, device drivers are supplied in the OS to access real physical memory as standard VME I/O. These may have names such as/dev/vme_24_16.

Some systems also want the DVME-601 memory reservation allocated at boot-up time in a system configuration file. You may also have to access the board as a logical "file" device using the ioctrl functions. Before you boot the OS, try reading and writing a few values into low DPR using your PROM Monitor/Debugger. This will confirm that address decoding is working. Remember that some device drivers expect the supervisor (privileged) mode for access.

7.8 Quiet Mode Conversion

With higher gain, longer signal leads or a high resolution converter, system noise may increase. The narrow aperture sample/hold picks off random portions of the combined signal and noise and faithfully digitizes the total signal.

Consider averaging a small array of A/D samples to smooth the noise. Once the MUX address is locked on one channel, no further settling delay is required and single-channel samples may be taken as fast as EOC appears. A suggested technique is to collect a power-of-2 number of samples, then do a log-2 shift right to do a quick average. (Example: take 8 samples then shift right 3 bits.) This avoids floating point averaging operations in assembly language.

Slightly lower system noise may also be available by making the microprocessor go to sleep while doing A/D conversion. The idea here is to prevent instruction fetches during this time. This reduces digital noise being radiated on the address and data buses.

One way to do this is to unmask the EOC interrupt in the 68901 controller and write a local EOC interrupt subroutine. After the last conversion is transferred to DPR, start the next conversion then TRAP the 68010 to supervisor state and STOP it. During the STOP, the 68010 executes internal NOP's. The EOC interrupt can be programmed to break the STOP.

Another way to do this without writing an interrupt is to decrement a delay counter during conversion. IF the loop is tight enough, all instructions remain inside the 68010's instruction queue and no external opcode fetches are made. After the previous EOC, advance the channel address if not already done in autoincrement (to get started settling). Next load the delay counter then read the A/D data register to start the next conversion. The A/D data register and DPR are indirectly addressed by registers A0 and A1 respectively:

```
        MOVE.W   #DELAY,D1
        MOVE.W   (A0), (A1)+  *read last sample, start next
LOOP:   DBF      D1,LOOP  *no opcode fetch during delay
```

After the delay, resume EOC polling again. Adjust the delay to be just slightly longer than the expected EOC time.

THEORY OF OPERATION

8.0 Repair and Servicing

The information presented here is not intended as a repair and troubleshooting guide. It is included in case you require better understanding of the DVME-601. This board is easily damaged by personnel who are not trained in its repair. Except for field recalibration, DATEL strongly recommends that the DVME-601 be returned to the home office for service.

The board is fabricated using multi-layer printed circuits, advanced hybrid microelectronics and precision custom test fixtures. Except for detection of gross failures, (such as a loss of 15 Volt dc power), the board should not be field repaired. Returning the board to the home office for repair will also provide the user with the latest software updates and analysis according to most recent Engineering revisions.

DATEL attempts to stock a small quantity of boards at its home office but cannot guarantee their availablity on short notice. Users with critical downtime applications should stock spare boards and test them periodically.

DATEL manufactures the proprietary data acquisition components on the DVME-601. Additional technical information on these devices is available on request.

8.1 Bus Interface Logic

(Refer to DATEL Drawing Number 15260 Sheet 1)

VME Data Bus lines D15 - D00 are buffered by octal transceivers U47 and U48 (74ALS645-1) to produce on-board data bus lines BD15 - BD00. VME Address Bus lines A15 - A01 are buffered and latched by octal registers U42 and U43 (74LS374). The upper Address Bus Lines A23 - A16 are connected to octal comparator U45 (74ALS518) and are compared to Base Address Jumpers 41 to 56 to select a 64 Kbyte address block. The Address Modifier lines are decoded by bipolar PROM U44 (TBP24S10) and produce a high signal at U34-3 on a valid AM code condition. PAL device U34 (PAL16L8) is used to decode VME Bus control lines DS1*, DS0*, AS*, WRITE*, and the base address and AM code select signals. The outputs of this PAL control the Address latches and data bus transceivers. This PAL is also used to set the Board Select F/F U31-19 (PAL20RA10). PAL device U33 (PAL16L8) decodes the latched VME address bus and generates Read and Write strobes to the Dual Port RAM and on-board registers. The I/O space for the registers overlays the high 8 bytes of the Dual Port RAM; this I/O space is decoded by NAND gate U30 (74S133). Shift Register U40 (74HCT4015) is used to provide a short delay for generating DTACK* to the VME Bus CPU. PAL device U32 (PAL20R4) is used to implement a VME Bus Interrupter circuit. The Interrupt Vector number may be written to octal latch U35 (74LS374) and is placed on the data bus during an interrupt cycle.

8.2 Dual Port RAM and Arbitration Logic

(Refer to DATEL Drawing Number 15260 Sheet 2)

The 32 kword Dual Port RAM (DPR) is composed of U22 and U23 (HM62256). The RAM address lines are connected to line driver IC's U20 (29C841), U21, and U29 (74ALS244). The local 68010 and VME Bus CPU each drive one set of line drivers, allowing each to access the DPR address lines. Likewise, the DPR data lines are buffered by octal tranceivers U8, U9, U24, and U25 (74ALS645-1). The DPR control lines are similarly buffered by line driver U7 (74ALS244).

Either the local 68010 CPU or the VME bus CPU may have access to the DPR at any given time; only one set of tranceivers is enabled, depending on which CPU has been granted access by the arbitration PAL U31 (PAL20RA10). The internal Local and VME flip-flops in this PAL are clocked on opposite phases of a clock,

and are used to grant DPR access when either the local CPU or VME CPU attempts to use the DPR. The local CPU's DTACK* is held off while the host has access to the DPR. Only one side can be selected at a time. Also, two flip flops in the PAL20RA10 are used as toggle flip flops to divide the 16 MHz input clock to 8 and 4 MHz clocks which are used by the MC68010 and MC68901, respectively.

8.3 MC68010 CPU and Memory Section

(Refer to DATEL Drawing Number 15260 Sheet 3)

The local CPU is a MC68010 microprocessor (U5). U18 and U19 (27256 or 27512) form a 32 or 64 kword program memory, while U27 and U28 (HM62256) form a 32 kword local RAM. PAL device U4 (PAL20S10) is used to combine DTACK* from the various I/O registers and transmit it to the MC68010. Additionally, the PAL is used to decode memory and register addresses.

8.4 MC68901 Multifunction Peripheral Section

(Refer to DATEL Drawing Number 15260 Sheet 4)

The MC68901 Multi Function Peripheral (U17) contains a USART, a general purpose I/O port and 4 independent timers. Timer B (and optionally Timer A) are used by Executive firmware for the A/D start trigger but may be reassigned by local user programs. Timer D is dedicated to the UART baud rate clock. Unused timers may be used for waveform generation, elapsed time measurement, etc. See the Motorola 68901 programming information. General Purpose I/O lines I7, I6, and I5 are committed to servicing interrupts from the VME Bus, EOC, and EOS. The remaining five lines are for general purpose use.

8.5 Command Register and Address Decode Logic

(Refer to DATEL Drawing Number 15260 Sheet 4)

The Command Register is composed of octal register U26 (74ALS273) and F/F U36-9. The Command Register may be written by the 68010 and is used to select the A/D conversion mode and to control the LED lamp. The PAL circuit U14 (PAL16L8) is used to decode the I/O register address for the MC68010. F/F U31-21 is part of the VME Bus Command register and may be written or read to enable interrupts from the MC68010 to the VME Bus.

8.6 A/D Converter Section

(Refer to DATEL Drawing Number 15260 Sheet 5)

The Start Channel Register (U11) and the Final Channel Register (U10) (74ALS273) may be written by the MC68010 CPU. The Current Channel counter U13 (PAL20RA10) is loaded with the contents of the Start Channel Register when the Start Channel register is loaded or at End of Scan. Octal Comparator U12 (74ALS518) is used to generate a Start = Final signal at pin 19. This signal is used to set the EOS F/F U37-5 (74LS74). PAL device U16 (PAL20L8) is used to multiplex the A/D start convert pulse from a number of sources depending on the Command Register contents. The Sample-Hold amplifier and A/D converter are contained in Module A1. Several different modules may be used to obtain different resolutions and conversion speeds. The EOC F/F U37-9 is set at the end of conversion and may be read by the MC68010 through line driver U38-9 (8T97). The Analog Multiplexer is composed of U2 (MX1616) and may be configured for 16 Single-Ended or 8 Differential input channels. The Instrumentation Amplifier is composed of U1 (AM551). Module PS1 is a DC to DC converter and produces ±15V for the linear circuits.

A/D SECTION CALIBRATION

9.0 Introduction

The DVME-601 is supplied calibrated and tested by DATEL. It should be recalibrated at 90 day intervals depending on its operating environment. To avoid damage, users performing calibration should be thoroughly familiar with electronic test equipment, the host computer and microelectronic assemblies. Handle the board very carefully with anti-static protection. Board calibration should be performed according to the following procedure:

9.1 Equipment Required

Voltage calibrator (DATEL DVC-8500 or equivalent).
Digital Voltmeter (Fluke 8800A or equivalent).
Dumb CRT terminal and RS-232-C cable for the J2 Monitor.
VME Extender board, miniature jeweler's screwdrivers and clipleads. Assembly and schematic drawings (optional) to verify jumpers, test points and pots.

9.2 Board Setup

Allow 20 minutes warmup for DVME-601F. The calibration environment should be electrically quiet. Avoid large industrial switching noise or RF fields during calibration. Use short leads to the signal input connections and do not route them near electrically noisy sources. Avoid other digital boards near the DVME-601 in the VMEbus chassis. Especially avoid the switching power supply.

It is not necessary to have a host program running to perform analog signal calibration on the DVME-601. The VMEbus will only be used to supply +5V dc, ±12V dc power and the 16 MHz SYSCLK signal. A/D conversion will be controlled by the serial port Monitor A/D command. However, a host program would be advisable to thoroughly check out A/D data transfers through the DPR.

Install the board in the VME host, connect the CRT terminal at J2, apply host power and confirm that the 601 is operating properly as described in earlier sections. Shut off host power, remove the 601 and replace it with a VME extender board. Make sure the extender carries all the bus signals needed by the 601. If not already done, set the board A/D jumpers for differential input operation and do not bypass the PGA. Install the 601 on the extender, connect the CRT terminal at J2, apply power and reconfirm that the 601 is operating normally.

Using the A/D command in the firmware Monitor, select Channel 0 (AD 0). Connect the calibrator to the Channel 0 inputs (J1 pins 24 and 12). Connect the LO side of the input to analog ground (J1 pin 12 to pin 13). The CRT terminal will show rapidly repeating A/D samples in hexadecimal output.

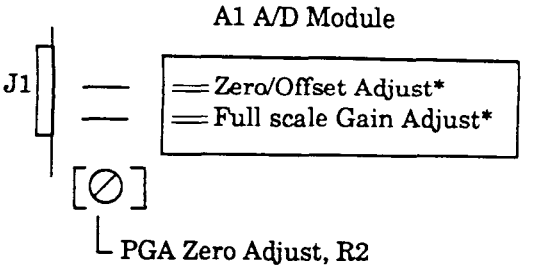
9.3 Instrumentation Amplifier Zero Adjust

Consult the assembly drawing and examine the board. The PGA zero adjustment potentiometer R2 is adjacent to the J1 connector. Set the calibrator for 0.0000V. Test point TP11 is the output of the PGA and the input to the A/D module. It is also connected to jumper post 9 and it may be necessary to partially withdraw the post plug to gain access. Or make contact on the non-component side of the board. Monitor TP11/post 9 with the DVM. Adjust the potentiometer R2 for a reading of 0.0000V.

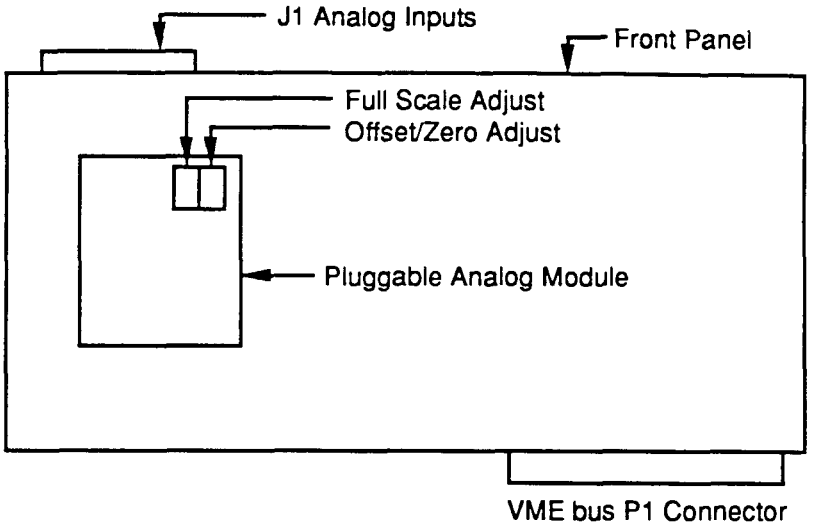
9.4 A/D Adjustments

The A/D converter module A1 is a precision encapsulated assembly. Models DVME-601A, B, E, and C use modules which contain two miniature multi-turn potentiometers to adjust the zero/offset and full scale gain. The DVME-601D is autozeroing and contains only a full scale pot. The A/D module pots are accessed

at the J1 end of the module, just above the U2 MX-1616 multiplexer as shown below in Figure 9.0:



*Note: On the DVME-601F, the innermost pot is gain and the outer pot is offset/zero on the ADC-14/2 module.



DVME-601F Trimpot Orientation

Figure 9.0 Calibration Pot Location

The two pots interact slightly, requiring repeated adjustments. The hexadecimal data displayed depends on the number of bits of resolution, whether the converter is unipolar or bipolar and whether coding is 2's complement or straight/offset binary. A/D data is left justified with zero bits filling the right (least significant) unused bits. For the 12-bit converters (601A, B, E), positive full scale will be displayed as FFF0 (hex). The 14-bit converter will display FFFC hex full scale. The 16-bit 601C or D shows FFFF hex full scale. Two's complement coding inverts the most significant bit showing 7FF0, 7FFC or 7FFF full scale respectively.

Bipolar converters use the most significant bit (MSB) to indicate polarity whereas unipolar converters imply the positive polarity and do not have a polarity bit in the data output.

The calibration procedure for all converters is essentially the same. Since the A/D is very linear, it only needs to be calibrated near its end points. Do not use the actual end points since we wish to observe some adjustment range. Make sure the calibrator and test leads do not introduce noise into the PGA inputs. The calibration voltages are chosen to be at the decision points midway between code states. It is normal for them to flicker slightly with each new conversion, especially for high resolution converters. Adjust the pots so that any flicker spans the code states equally.

For the 14 and 16-bit A/D converters, it may be necessary to use precision attenuators or vernier outputs on the calibrator to achieve the sub-millivolt outputs.

The zero/offset pot adjusts either the zero for unipolar converters or the minus full scale (offset) for bipolar converters. The full scale pot is always for positive full scale. In the tables below, calibration points are given for additional values besides zero and full scale in case you wish to verify linearity at the carry points of each bit. Major carries (example: EFF0 <-> F000) may be somewhat difficult to interpret on the terminal.

If you wish to select a fixed higher gain on the PGA by adding a gain resistor, the calibration procedures are the same but use smaller input voltages which you will have to derive. Data output flicker may increase with gain.

The bit weight ("LSB") is referenced in the discussions below. Bit weight is the amount of voltage difference necessary to cause a transition between two code states in an ideal A/D converter. It is the total input voltage span (both positive and negative portions) divided by the number of code states which equal 2 to the power of the number of resolution bits minus 1. It may be derived as follows:

Bit weight = Input span (Volts)/((2 ^ (no. resolution bits)) - 1)

9.5 Zero/Offset Adjust

With the A/D converting on Channel 0, apply -Full Scale + 1/2 LSB to the Channel 0 input. The inputs to be used are shown in Figure 9.1. For 12-bit converters, adjust the Zero/Offset pot in the A/D module so that the output data is fluctuating between 0000 and 0010 hex for binary coding or 8000 and 8010 hex for 2's complement. For the 14-bit converter, the codes are 0000 and 0004 or 8000 and 8004. For 16 bits, the codes are 0000 and 0001 or 8000 and 8001. See Figure 9.2.

	-Full Scale + 1/2 LSB		
Range	**12-bit A/D**	**14-bit A/D**	**16-bit A/D**
0 to +5V	+ 0.0006V	NA	NA
0 to +10V	+ 0.0012V	NA	+0.0002V
±5V	- 4.9988V	-4.9997V	NA
±10V	- 9.9976V	-9.9994V	-9.9999V

NA = Not available for this model.

Figure 9.1 Zero or Offset Calibration Inputs

	12-bit A/D	14-bit A/D	16-bit A/D
Coding	**Adjust for output between:**		
	0010	0004	0001
Binary	0000	0000	0000
Two's	8010	8004	8001
Compl.	8000	8000	8000

Figure 9.2 Data Outputs for Zero or Offset Adjust

9.6 A/D Gain Adjustment

With the A/D converting on Channel 0, apply +Full Scale -1.5 LSB to the Channel 0 input as shown in Figure 9.3. For the 12-bit A/D converter, adjust the full scale Gain pot in the A/D module to observe A/D readings fluctuating between FFE0 hex and FFF0 for binary coding or 7FE0 and 7FF0 hex for 2's complement. For the 14-bit converter, the codes are FFF8 and FFFC or 7FF8 and 7FFC. For the 16 bit converter, the codes are FFFE and FFFF or 7FFE and 7FFF.

The following are the voltage inputs for the full scale gain calibration:

	+Full Scale -1.5 LSB		
Range	**12-bit A/D**	**14-bit A/D**	**16-bit A/D**
0 to +5V	+4.9982V	NA	NA
0 to +10V	+9.9963V	NA	+9.9997V
±5V	+4.9963V	+4.9991V	NA
±10V	+9.9926V	+9.9982V	+9.9995V

Figure 9.3 Full Scale Gain Calibration Inputs

Coding	**12-bit A/D Adjust for output between:**	**14-bit A/D**	**16-bit A/D**
Binary	FFF0 FFE0	FFFC FFF8	FFFF FFFE
Two's Compl.	7FF0 7FE0	7FFC 7FF8	7FFF 7FFE

Figure 9.4 Data Outputs for Full Scale Gain Adjust

Once the gain adjustment is done, repeat the zero/offset test again. Since these two adjustments interact, repeat them until no further improvement can be obtained.

This completes the analog calibration for the Programmable Gain Amplifier and A/D converter. A more complete calibration table for 12-bit A/D converters is shown in Figure 9.5. It may be used to check accuracy at each bit carry. The output code should agree within the accuracy and linearity error specification. For process control applications, the equivalent voltage input is listed for a 4 to 20 mA input using an external 250 ohm shunt, mounted on the DATEL DVME-691 terminator panel or the DVME-641 slave expansion MUX board.

Unipolar			Bipolar		BIN Code (hex)	2's C Code (hex)
0 - 5V	0 - 10V	4-20mA	±5V	±10V		
0.0000	0.0000	1.0000	-5.0000	-10.0000	0000	8000
0.0012	0.0024	1.0010	-4.9976	-9.9951	0010	8010
0.0024	0.0049	1.0020	-4.9951	-9.9902	0020	8020
0.0049	0.0098	1.0039	-4.9902	-9.9805	0040	8040
0.0098	0.0196	1.0078	-4.9805	-9.9609	0080	8080
0.0196	0.0391	1.0156	-4.9609	-9.9219	0100	8100
0.0391	0.0781	1.0312	-4.9219	-9.8437	0200	8200
0.0781	0.1563	1.0625	-4.8437	-9.6875	0400	8400
0.1563	0.3125	1.1250	-4.6875	-9.3750	0800	8800
0.3125	0.6250	1.2500	-4.3750	-8.7500	1000	9000
0.6250	1.2500	1.5000	-3.7500	-7.5000	2000	A000
1.2500	2.5000	2.0000	-2.5000	-5.0000	4000	C000
2.5000	5.0000	3.0000	0.0000	0.0000	8000	0000
3.7500	7.5000	4.0000	2.5000	5.0000	C000	4000
4.3750	8.7500	4.5000	3.7500	7.5000	E000	6000
4.6875	9.3750	4.7500	4.3750	8.7500	F000	7000
4.8437	9.6875	4.8750	4.6875	9.3750	F800	7800
4.9219	9.8437	4.9375	4.8437	9.6875	FC00	7C00
4.9609	9.9219	4.9688	4.9219	9.8437	FE00	7E00
4.9805	9.9609	4.9844	4.9609	9.9219	FF00	7F00
4.9902	9.9805	4.9922	4.9805	9.9609	FF80	7F80
4.9951	9.9902	4.9961	4.9902	9.9805	FFC0	7FC0
4.9976	9.9951	4.9980	4.9951	9.9902	FFE0	7FE0
4.9988	9.9976	4.9990	4.9976	9.9951	FFF0	7FF0

Figure 9.5 12-Bit A/D Calibration Table

9.7 DVME-601 Code

/* SCAN22A.C

This program shows a simple scan to the DPR. Once the DPR is filled, it may be viewed using the 601 serial port Monitor command MD 40000,100 or using the host debugger or use the printf's at the end of main ( ).

This code executed properly on a Force VME system running PDOS, although no explicit calls to PDOS are used. This "C" code should be highly portable. Just remember to change the base address (BASPTR) for your system. The intent here is to show a modular way of programming the 601 using functions such as exec_on ( ), load (AFBptr, length), go ( ), buflrdy ( ), chkerr ( ), etc. Another goal is to show that 601 programming is not difficult.

We have implemented a very flexible Application Function Block loader here with several good features:

1. The 32-bit longwords are loaded symbolically with #defined SYMBOLS which can be extended or modified at the beginning of the code. Subroutine names from the SUBDEF.C file are also loaded symbolically.

2. The AFB may be easily edited in the "C" source code in case you wish to change the functions around. The idea is that an AFB is just a variable-length array such that the load ( ) function figures out the length (and therefore how much to load) from the sizeof (array) operator.

3. Finally, we must load either bytes or 16-bit words since the 601 cannot transfer 32-bit longwords. The load process must accomodate the decrementing longword structure of an AFB.

Error handling could be improved by using a more global error decoder. In the program presented here, we poll for awhile, report the error, then exit. The alternate choice is to not report the error but simply return a code or set a global error value. The EXTRIG.C program (by a different programmer) shows a more complete approach. We have used the convention of returning a zero if there is no error or if the condition is true.

Old style initialization is used for PDOS "C".
Remember to change BASPTR! Code begins:
*/

```
#include <stdio.h>
/* #include <SUBDEF.C> This file is on the 601 distribution disk. */

#define EXEC 0x45584543 /* "EXEC" ASCII string */
#define SPONCR 0x204F4E0D /* space "ON" cr */
#define SPOFF 0x204F4646 /* space "OFF" */
#define CRZERO 0x0D000000 /* cr, 3 zero's */
#define BASPTR 0xFA0000 /* board BASE address - change for your system!! */
#define BEG0 0x42454730 /* DPR ASCII markers & function cmds */
#define BEG1 0x42454731
#define STRS 0x53545253
#define ENDC 0x454E4443
#define ENDF 0x454E4446
#define CONT 0x434F4E54

#define dfsbuf 0x0000012E /* EPROM A/D subroutines - see manual */
#define stmlsc 0x0000018E
```

```
/* externals usable by several functions: */

static unsigned char *exeptr, *cmdptr, *bas_ptr;
/* cmdptr is a gen'l purpose AFB ptr */
static unsigned short *dprptr, *intr601, *adstat, *hststa, *datptr;
char go ( ), buf1rdy ( ), buf2rdy ( ), chkerr ( ), execon ( ), execoff ( ), fnc0done ( ), fnc1done ( );
int cntr;

main ( ) {
  int sza, szb; /* array sizeof variables */

  /* Application Function Blocks - these may be edited for desired subroutine sequence */

  static unsigned long afba [ ] = {
    BEG0,
    dfsbuf, /* define buffer */
      0,
      64,
    ENDF,
    STRS,
    ENDC
  };
  static unsigned long afbb [ ] = {
    BEG0,
    stmlsc, /* A/D scan subroutine called from DPR */
      0, /* start channel address */
      3, /* final channel address */
      8, /* number of scans */
    ENDF,
    STRS,
    ENDC
  };

  sza = sizeof (afba);
  szb = sizeof (afbb);

  /* pointers & offsets to DPR locations. Note (casts): */
  bas_ptr = BASPTR;
  dprptr = (unsigned char *) (BASPTR + 0xFFEC);
  intr601 = (unsigned short *) (BASPTR + 0xFFFC);
  adstat = (unsigned short *) (BASPTR + 0xFFF4);
  hststa = (unsigned short *) (BASPTR + 0xFFF6);
```

```
  execon ( );
  load (afba, sza);
  if ( go ( ) ! = 0 ) {
     printf ("\nACK not rcvd loading AFBa.");
     exit (0);
  }
  printf ("\nAFBa loaded.  Buf 1 defined.");
  if ( fnc0done ( ) ! = 0 ) {
     printf ("\nFunction 0 failed.");
     exit (0);
  }
  else
     printf ("\nFunction 0 done.");
  shosta ( );
  chkerr ( );
  printf ("\nStarting scan AFB.");
  load (afbb,szb);
  if ( go ( ) ! = 0) {
     printf ("\nACK not rcvd loading AFBb");
     shosta ( );
     exit (0);
  }
  chkerr ( );
  shosta ( );
  printf ("\nScan AFB running.");
  for (cntr=1000 ; cntr > 0 ; cntr-- ) {
     if  ( buf1rdy ( ) == 0) {
        printf ("\nGot bufrdy.");
        shosta ( );
        break;
     }
  }
  if (cntr == 0) {
     printf ("\nA/D Buffer not ready.");
     shosta ( );
     exit (0);
  }
  printf ("\nBuffer filled with A/D data.");
  datptr = bas_ptr;
  shodat (datptr, 32);
  execoff ( );
} /*  end of main  */
```

```
/*                          FUNCTION LIBRARY                          */

char execon ( ) { /* turns EXEC ON */
  unsigned char i;
  /* warning - the following array may load incorrectly for 80x86 hosts with reverse byte ordering */
  static unsigned long exe [ ] = {
    EXEC,
    SPONCR
  };
  exeptr = (unsigned char *) (BASPTR + 0xFFE6);
  cmdptr = exe; /* note pointer type mismatch - compiles OK */
  for (i=0; i <sizeof (exe) ; i++)
    *exeptr++ = *cmdptr++;
  if ( go ( ) ! = 0 ) {
    printf ("\nHSTSTA ACK failed.");
    shosta ( );
    exit (0);
  }
  if ( ( *adstat & 0x8000 ) ! = 0x8000 | |
    ( *hststa & 0x0001 ) ! = 0x0001 ) {
    printf ("\nEXEC ON fail");
    shosta ( );
    exit (0);
  }
  printf ("\nEXEC is ON.");
  return 0;
}

char execoff ( ) { /* turns EXEC OFF */
  unsigned char i;
  /* warning - the following array may load incorrectly for 80x86 hosts with reverse byte ordering */
  static unsigned long exe [ ] = {
    EXEC,
    SPOFF,
    CRZERO
  };
  exeptr = (unsigned char *) (BASPTR + 0xFFE6);
  cmdptr = exe; /* note pointer type mismatch - compiles OK */
  for (i=0; i < sizeof (exe) ; i++)
    *exeptr++ = *cmdptr++;
  if ( go ( ) ! = 0 ) {
     printf ("\nHSTSTA ACK failed.");
    shosta ( );
     exit (0);
  }
  if ( ( *adstat & 0x8000 ) ! = 0 | |
    ( *hststa & 0x0001 ) ! = 0x0001 ) {
    printf ("\nEXEC OFF fail");
    shosta ( )
    exit (0);
  }
  printf ("\nEXEC is OFF.");
  return 0;
}
```

```
/* go ( ) resets HSTSTA error bits & ACK, interrupts 601, polls ACK, returns ACK status. */

char go ( ) {
  unsigned short cntr;
  *hststa = 0; /* reset ACK and error bits */
  *intr601 = 0; /* interrupt 601 */
  for (cntr = 40000 ; cntr > 0 ; cntr-- ) {
    if ( ( *hststa & 0x0001 ) == 0x0001) /* ACK set ? */
      break;
  }
  if (cntr > 0)
    return 0; /* rcvd ACK OK */
  return 1; /* ACK fail */
}

shosta ( ) { /* show status words */
  printf ("\nADSTAT = $%04x, HSTSTA = $%04x", *adstat, *hststa );
}

load (afbptr,sz) /* loads an AFB at BASE + FFEC. Needs go ( ) to start */
int sz; /* bytes in AFB array */
unsigned long *afbptr; {
  unsigned char *loadptr, i;
  cmdptr = dprptr;
  loadptr = afbptr; /* pointer mismatch compiles OK */
  while (sz > 0) {
    for (i = 0; i < 4; i++ ) {
      *cmdptr++ = *loadptr++;
      sz--;
    }
    cmdptr = cmdptr - 8; /* walk down the list in longwrds */
  }
}

char buf1rdy ( ) { /* If buf 1 rdy, resets rdy bit and rtns 0 */
  unsigned short temp;
  temp = *adstat;
  if ( ( 0x0002 & temp ) == 0x0002 ) {
    *adstat = temp & 0xfffd; /* reset rdy bit */
    return 0;
  }
  return 1; /* buf 1 not rdy */
}
```

```
char buf2rdy ( ) { /* If buf 2 rdy, resets rdy bit and rtns 0 */
  unsigned short temp;
  temp = *adstat;
  if ( ( 0x0004 & temp ) == 0x0004 ) {
    *adstat = temp & 0xfffb; /* reset rdy bit */
    return 0;
  }
  return 1; /* buf 2 not rdy */
}

/* chkerr ( ) checks the HSTSTA error bits. Use this routine if data ready does not return in reasonable
time or other suspicious delays. */

char chkerr ( ) {
  if ( ( 0x003e & *hststa) ! = 0 ) {
    printf ("\nHSTSTA error.");
    shostsa ( );
    return 1;
  }
  return 0; /* no errors */
}

shodat (datptr, wds) /* prints data at datptr */
unsigned short *datptr; /* array start addr */
int wds; { /* words in array */
  int i;
  printf ("\n");
  while (wds > 0) {
    printf ("\nData at $%1x:\n", datptr);
    for ( i=0; i < 8 ; i++ ) {
      printf ("$%04x", *datptr++);
      wds--;
    }
  }
}
char fnc0done ( ) { /* tells if function 0 is done */
  if ( ( 0x0040 & *hststa ) == 0x0040 )
    return 0;
  return 1;
}

char func1done ( ) { /* tells if function 1 is done */
  if ( ( 0x0080 & *hststa ) == 0x0080 )
    return 0;
  return 1;
}
/* end of program */
```