MVME177

Single Board Computer

Installation and Use Manual

VME177A/IH2

Notice

While reasonable efforts have been made to assure the accuracy of this document, Motorola, Inc. assumes no liability resulting from any omissions in this document, or from the use of the information obtained therein. Motorola reserves the right to revise this document and to make changes from time to time in the content hereof without obligation of Motorola to notify any person of such revision or changes.

It is possible that this publication may contain reference to, or information about Motorola products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that Motorola intends to announce such Motorola products, programming, or services in your country.

Restricted Rights Legend

If the documentation contained herein is supplied, directly or indirectly, to the U.S. Government, the following notice shall apply unless otherwise agreed to in writing by Motorola, Inc.

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Motorola, Inc.
Computer Group
2900 South Diablo Way
Tempe, Arizona 85282

Preface

The *MVME177 User's Manual* provides general information, hardware preparation and installation instructions, operating instructions, and functional description for the MVME177 Single Board Computer (referred to as MVME177 throughout this manual). The information contained in this manual applies to the following MVME177 models:

MVME177-001	MVME177-011
MVME177-002	MVME177-012
MVME177-003	MVME177-013
MVME177-004	MVME177-014
MVME177-005	MVME177-015
MVME177-006	MVME177-016

This manual is intended for anyone who wants to design OEM systems, supply additional capability to an existing compatible system, or work in a lab environment for experimental purposes.

A basic knowledge of computers and digital logic is assumed.

To use this manual, you should be familiar with the publications listed in the *Related Documentation* section in Chapter 1 of this manual.

The computer programs stored in the Read Only Memory of this device contain material copyrighted by Motorola Inc., first published 1990, and may be used only under a license such as the License for Computer Programs (Article 14) contained in Motorola's Terms and Conditions of Sale, Rev. 1/79.

All Motorola PWBs (printed wiring boards) are manufactured by UL-recognized manufacturers, with a flammability rating of 94V-0.

This equipment generates, uses, and can radiate electromagnetic energy. It may cause or be susceptible to electromagnetic interference (EMI) if not installed and used in a cabinet with adequate EMI protection.

CE European Notice: Board products with the CE marking comply with the EMC Directive (89/336/EEC). Compliance with this directive implies conformity to the following European Norms:

EN55022 (CISPR 22) Radio Frequency Interference

EN50082-1 (IEC801-2, IEC801-3, IEEC801-4) Electromagnetic Immunity

The product also fulfills EN60950 (product safety) which is essentially the requirement for the Low Voltage Directive (73/23/EEC).

This board product was tested in a representative system to show compliance with the above mentioned requirements. A proper installation in a CE-marked system will maintain the required EMC/safety performance.

Printed in the United States of America

June 1996

Safety Summary
Safety Depends On You

The following general safety precautions must be observed during all phases of operation, service, and repair of this equipment. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the equipment. Motorola, Inc. assumes no liability for the customer's failure to comply with these requirements.

The safety precautions listed below represent warnings of certain dangers of which Motorola is aware. You, as the user of the product, should follow these warnings and all other safety precautions necessary for the safe operation of the equipment in your operating environment.

Ground the Instrument.

To minimize shock hazard, the equipment chassis and enclosure must be connected to an electrical ground. The equipment is supplied with a three-conductor ac power cable. The power cable must be plugged into an approved three-contact electrical outlet. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

Do Not Operate in an Explosive Atmosphere.

Do not operate the equipment in the presence of flammable gases or fumes. Operation of any electrical equipment in such an environment constitutes a definite safety hazard.

Keep Away From Live Circuits.

Operating personnel must not remove equipment covers. Only Factory Authorized Service Personnel or other qualified maintenance personnel may remove equipment covers for internal subassembly or component replacement or any internal adjustment. Do not replace components with power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

Do Not Service or Adjust Alone.

Do not attempt internal service or adjustment unless another person capable of rendering first aid and resuscitation is present.

Use Caution When Exposing or Handling the CRT.

Breakage of the Cathode-Ray Tube (CRT) causes a high-velocity scattering of glass fragments (implosion). To prevent CRT implosion, avoid rough handling or jarring of the equipment. Handling of the CRT should be done only by qualified maintenance personnel using approved safety mask and gloves.

Do Not Substitute Parts or Modify Equipment.

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification of the equipment. Contact your local Motorola representative for service and repair to ensure that safety features are maintained.

Dangerous Procedure Warnings.

Warnings, such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed. You should also employ all other safety precautions which you deem necessary for the operation of the equipment in your operating environment.

WARNING

Dangerous voltages, capable of causing death, are present in this equipment. Use extreme caution when handling, testing, and adjusting.

Contents

List of Figures

List of Tables

General Information 1

Introduction

This manual provides:

- General information
- Preparation for use and installation instructions
- Operating instructions
- Functional description

for the MVME177 series of Single Board Computers (referred to as the MVME177 throughout this manual).

Model Designations

The MVME177 is available in the models listed in Table 1 - 1.

Table 1-1. MVME177 Model Designations

Model Number	Speed	Major Differences
MVME177-001	50 MHz	MC68060, 4MB Onboard ECC DRAM
MVME177-002	50 MHz	MC68060, 8MB Onboard ECC DRAM
MVME177-003	50 MHz	MC68060, 16MB Onboard ECC DRAM
MVME177-004	50 MHz	MC68060, 32MB Onboard ECCDRAM
MVME177-005	50 MHz	MC68060, 64MB Onboard ECC DRAM
MVME177-006	50 MHz	MC68060, 128MB Onboard ECC DRAM
MVME177-011	60 MHz	MC68060, 4MB Onboard ECC DRAM
MVME177-012	60 MHz	MC68060, 8MB Onboard ECC DRAM
MVME177-013	60 MHz	MC68060, 16MB Onboard ECC DRAM
MVME177-014	60 MHz	MC68060, 32MB Onboard ECCDRAM
MVME177-015	60 MHz	MC68060, 64MB Onboard ECC DRAM
MVME177-016	60 MHz	MC68060, 128MB Onboard ECC DRAM

Features

Features of the MVME177 are listed in the following table:

Table 1-2. MVME177 Features

Feature	Description
Microprocessor	MC68060 at 50 MHz (MVME177-00*x*) or 60 MHz (MVME177-01*x*)
DRAM	4/8/16/32/64/128/256MB with ECC protection
Flash Memory	4MB in four Intel 28F008SA chips with software control write protection
EPROM	1MB in two 44-pin PLCC sockets (organized as one bank of 32 bits)
Jumper and software control	Mixed EPROM/Flash, or All Flash configuration
SRAM	128KB (with optional battery backup)
Status LEDs	Eight LEDs: for FAIL, STAT, RUN, SCON, LAN, +12V (LAN power), SCSI, and VME.
RAM	8K by 8 RAM and time of day clock with battery backup
Switches	RESET
	ABORT
Tick timers	Four 32-bit tick timers for periodic interrupts
Watchdog timer	One watchdog timer
Software interrupts	Eight software interrupts
I/O	SCSI Bus interface with DMA
	Four serial ports with EIA-232-D buffers with DMA
	8-bit bidirectional parallel port
	Ethernet transceiver interface with DMA
VMEbus interface	VMEbus system controller functions
	VMEbus interface to local bus (A24/A32, D8/ D16/D32 (D8/D16/D32/D64BLT) (BLT = Block Transfer)
	Local bus to VMEbus interface (A16/A24/A32, D8/D16/D32)
	VMEbus interrupter
	Global CSR for interprocessor communications
	DMA for fast local memory - VMEbus transfers (A16/A24/A32, D16/D32 (D16/D32/D64BLT)
Remote connector	For RESET and ABORT switches and LEDs

Specifications

General specifications for the MVME177 are listed in Table 1-3.

The following sections detail cooling requirements and FCC compliance.

Cooling Requirements

The Motorola MVME177 VMEmodule is specified, designed, and tested to operate reliably with an incoming air temperature range from 0˚ to 55˚ C (32˚ to 131˚ F) with forced air cooling at a velocity typically achievable by using a 100 CFM axial fan. Temperature qualification is performed in a standard Motorola VMEsystem chassis. Twenty-five watt load boards are inserted in two card slots, one on each side, adjacent to the board under test, to simulate a high power density system configuration. An assembly of three axial fans, rated at 100 CFM per fan, is placed directly under the VME card cage. The incoming air temperature is measured between the fan assembly and the card cage, where the incoming airstream first encounters the module under test. Test software is executed as the module is subjected to ambient temperature variations. Case temperatures of critical, high power density integrated circuits are monitored to ensure component vendors specifications are not exceeded.

While the exact amount of airflow required for cooling depends on:

- Ambient air temperature
- Type of board
- Number of boards
- Location of boards
- Other heat sources

adequate cooling can usually be achieved with 10 CFM and 490 LFM flowing over the module. Less airflow is required to cool the module in environments having lower maximum ambients. Under

more favorable thermal conditions, it may be possible to operate the module reliably at higher than 55˚ C with increased airflow. It is important to note that there are several factors, in addition to the rated CFM of the air mover, which determine the actual volume and speed of air flowing over a module.

Forced air cooling is required for the Atlas motherboard. Additional cooling is required with the installation of the MPC604 RISC processor. A 3-pin header (J17) is provided on the motherboard for powering a dedicated fan. Refer to the *Cooling Requirements* section in the *General Information* chapter for temperature qualification information for the system board platform.

Table 1-3. MVME177 Specifications

Characteristics	Specifications
Power requirements (with both EPROM sockets populated and excluding external LAN transceiver)	+5 Vdc (± 5%), 4.5 A (typical), 6.0 A (max.) (at 50 MHz, with 128MB ECC DRAM) +12 Vdc (± 5%), 100 mA (max.) (1.0 A (max.) with offboard LAN transceiver) -12 Vdc (± 5%), 100 mA (max.)
Operating temperature (refer to *Cooling Requirements* section)	0˚ to 55˚ C at point of entry of forced air (approximately 490 LFM)
Storage temperature	-40˚ to +85˚ C
Relative humidity	5% to 90% (non-condensing)
Physical dimensions PC board with mezzanine module only Height Depth Thickness PC boards with connectors and front panel Height Depth Thickness	Double-high VMEboard 9.187 inches (233.35 mm) 6.299 inches (160.00 mm) 0.662 inches (16.77 mm) 10.309 inches (261.85 mm) 7.4 inches (188 mm) 0.80 inches (20.32 mm)

FCC Compliance

The MVME177 was tested in an FCC-compliant chassis, and meets the requirements for Class A equipment. FCC compliance was achieved under the following conditions:

1. Shielded cables on all external I/O ports.
2. Cable shields connected to earth ground via metal shell connectors bonded to a conductive module front panel.
3. Conductive chassis rails connected to earth ground. This provides the path for connecting shields to earth ground.
4. Front panel screws properly tightened.

For minimum RF emissions, it is essential that the conditions above be implemented; failure to do so could compromise the FCC compliance of the equipment containing the module.

General Description

The MVME177 is a double-high VMEmodule based on the MC68060 microprocessor. The MVME177 has:

- 4/8/16/32/64/128/256 MB of ECC-protected DRAM
- 8KB of static RAM and time of day clock (with battery backup)
- Ethernet transceiver interface
- Four serial ports with EIA-232-D interface
- Four tick timers
- Watchdog timer
- 4 MB of Flash memory
- Two EPROM sockets
- SCSI bus interface with DMA

- One parallel port
- A 16/A24/A32/D8/D16/D32/D64 VMEbus master/slave interface
- 128KB of static RAM (with optional battery backup), and VMEbus system controller.

The I/O on the MVME177 is connected to the VMEbus P2 connector. The main board is connected through a P2 transition board and cables to the transition boards. The MVME177 supports the following transition boards:

- MVME712-12
- MVME712-13
- MVME712M
- MVME712A
- MVME712AM
- MVME712B

(referred to in this manual as MVME712*x*, unless separately specified).

The MVME712*x* transition boards provide configuration headers and industry standard connectors for the I/O devices.

The VMEbus interface is provided by an ASIC called the VMEchip2. The VMEchip2 includes:

- Two tick timers
- A watchdog timer
- Programmable map decoders for the master and slave interfaces
- VMEbus to/from local bus DMA controller
- VMEbus to/from local bus non-DMA programmed access interface

- VMEbus interrupter
- VMEbus system controller
- VMEbus interrupt handler
- VMEbus requester

Processor-to-VMEbus transfers can be:

- D8
- D16
- D32

VMEchip2 DMA transfers to the VMEbus, however, can be:

- D16
- D32
- D16/BLT
- D32/BLT
- D64/MBLT

The PCCchip2 ASIC provides:

- Two tick timers
- Interface to the LAN chip
- SCSI chip
- Serial port chip
- Parallel (printer) port
- BBRAM

The MCECC memory controller ASIC provides the programmable interface for the ECC-protected DRAM mezzanine board.

Equipment Required

The following equipment is required to make a complete system using the MVME177:

- Terminal
- Disk drives and controllers
- One of the following Transition modules:
  - MVME712-12
  - MVME712-13
  - MVME712M
  - MVME712A
  - MVME712AM
  - MVME712B
- Connecting cables
- P2 adapter
- Operating system

The MVME177Bug debug monitor firmware (177Bug) is provided in the two EPROMs in sockets on the MVME177 main module. It provides:

- Over 50 debug, up/downline load, and disk bootstrap load commands
- Full set of onboard diagnostics
- One-line assembler/disassembler

177Bug includes a user interface which accepts commands from the system console terminal. 177Bug can also operate in a System Mode, which includes choices from a service menu. Refer to the *177Bug Diagnostics User's Manual* and the *Debugging Package for Motorola 68K CISC CPUs User's Manual* for details.

The MVME712*x* series of transition modules provide the interface between the MVME177 module and peripheral devices. They connect the MVME177 to:

- EIA-232-D serial devices
- Centronics-compatible parallel devices
- SCSI devices
- Ethernet devices

The MVME712*x* series work with cables and a P2 adapter.

Software available for the MVME177 includes:

- SYSTEM V/68
- Real-time operating systems
- Programming languages
- Other tools and applications

Contact your local Motorola sales office for more details.

Related Documentation

The following publications are applicable to the MVME177 and may provide additional helpful information. If not shipped with this product, they may be purchased by contacting your local Motorola sales office. Non-Motorola documents may be purchased from the sources listed.

Note Although not shown in the following list, each Motorola Computer Group manual publication number is suffixed with characters which represent the type and revision level of the document, such as "/*xx*2" (the second revision of a manual); a supplement bears the same number as a manual but has a suffix such as "/*xx*2A1" (the first supplement to the second revision of the manual).

Document Title	Motorola Publication Number
177Bug Diagnostics User's Manual	V177DIAA/UM
Debugging Package for Motorola 68K CISC CPUs User's Manual	68KBUG1/D and 68KBUG2/D
Single Board Computers SCSI Software User's Manual	SBCSCSI/D
Single Board Computers Programmer's Reference Guide	VMESBCA/PG1 and VMESBCA/PG2
MVME712M Transition Module and P2 Adapter Board User's Manual	MVME712M/D
MVME712-12, MVME712-13, MVME712A, MVME712AM, and MVME712B Transition Module and LCP2 Adapter Board User's Manual	MVME712A/D
M68060 Microprocessor User's Manual	M68060UM

The following publications are available from the sources indicated:

Versatile Backplane Bus: VMEbus, ANSI/IEEE Std 1014-1987, The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017 (VMEbus Specification). This is also available as *Microprocessor system bus for 1 to 4 byte data, IEC 821 BUS*, Bureau Central de la Commission Electrotechnique Internationale; 3, rue de Varembé, Geneva, Switzerland.

ANSI Small Computer System Interface-2 (SCSI-2), Draft Document X3.131-198X, Revision 10c; Global Engineering Documents, P.O. Box 19539, Irvine, CA 92714.

CL-CD2400/2401 Four-Channel Multi-Protocol Communications Controller Data Sheet, order number 542400-003; Cirrus Logic, Inc., 3100 West Warren Ave., Fremont, CA 94538.

82596CA Local Area Network Coprocessor Data Sheet, order number 290218; and *82596 User's Manual*, order number 296853; Intel Corporation, Literature Sales, P.O. Box 58130, Santa Clara, CA 95052-8130.

NCR 53C710 SCSI I/O Processor Data Manual, order number NCR53C710DM; and *NCR 53C710 SCSI I/O Processor Programmer's Guide*, order number NCR53C710PG; NCR Corporation, Microelectronics Products Division, Colorado Springs, CO.

MK48T08 Timekeeper ™ and 8Kx8 Zeropower ™ RAM data sheet in Static RAMs Databook, SGS-THOMPSON Microelectronics Group; North & South American Marketing Headquarters, 1000 East Bell Road, Phoenix, AZ 85022-2699.

DS1643 Nonvolatile Timekeeping RAM, Dallas Semiconductor Data Manual, 4401 South Beltwood Parkway, Dallas, Texas 75244-3292.

Support Information

You can obtain connector interconnect signal information, parts lists, and schematics for the MVME177 free of charge by contacting your local Motorola sales office.

Manual Terminology

Throughout this manual, a convention is used which precedes data and address parameters by a character identifying the numeric format as follows:

$	dollar	specifies a hexadecimal character
%	percent	specifies a binary number
&	ampersand	specifies a decimal number

Unless otherwise specified, all address references are in hexadecimal.

An asterisk (*) following the signal name for signals which are level significant denotes that the signal is true or valid when the signal is low.

An asterisk (*) following the signal name for signals which are edge significant denotes that the actions initiated by that signal occur on high to low transition.

In this manual, assertion and negation are used to specify forcing a signal to a particular state. In particular, assertion and assert refer to a signal that is active or true; negation and negate indicate a signal that is inactive or false. These terms are used independently of the voltage level (high or low) that they represent.

Data and address sizes are defined as follows:

- A byte is eight bits, numbered 0 through 7, with bit 0 being the least significant
- A word is 16 bits, numbered 0 through 15, with bit 0 being the least significant
- A longword is 32 bits, numbered 0 through 31, with bit 0 being the least significant

Hardware Preparation and Installation

2

Introduction

This chapter provides the following for the MVME177:

- Unpacking instructions
- Hardware preparation
- Installation instructions

The MVME712*x* transition module hardware preparation is provided in separate manuals. Refer to *Related Documentation* in Chapter 1.

Unpacking Instructions

Note If the shipping carton is damaged upon receipt, request carrier's agent be present during unpacking and inspection of equipment

Unpack equipment from shipping carton. Refer to packing list and verify that all items are present. Save packing material for storing and reshipping of equipment.

Caution

Avoid touching areas of integrated circuitry; static discharge can damage circuits.

2

Overview of Start-up Procedure

The following list identifies the things you will need to do before you can use this board, and where to find the information you need to perform each step. Be sure to read this entire chapter and read all Caution notes before beginning.

Table 2-1. Start-up Overview

What you will need to do ...	Refer to ...	On page ...
Set jumpers on your MVME177 module.	***Hardware Preparation***	2-4
Ensure that EPROM devices are properly installed in the sockets.	***Hardware Preparation***	2-4
Install your MVME177 module in the chassis.	***Installation Instructions***	2-12
Set jumpers on the transition board; connect and install the transition board, P2 adapter module, and optional SCSI device cables.	The user's manual you received with your MVME712 module, listed in ***Related Documentation***	1-9
	You may also wish to obtain the ***Single Board Computer SCSI Software User's Manual***, listed in ***Related Documentation***	1-9
Connect a console terminal to the MVME712.	***Installation Instructions***	2-12
	The user's manual you received with your MVME712 module, listed in ***Related Documentation***	1-9
Connect any other optional devices or equipment you will be using.	The user's manual you received with your MVME712 module, listed in ***Related Documentation***	1-9
	EIA-232-D Interconnections	A-1
	Port Numbers	B-34
	Disk/Tape Controller Data	C-1
Power up the system.	***Installation Instructions***	2-12
	Front Panel Indicators (DS1 - DS4)	3-3
	Troubleshooting the MVME177; Solving Start-up Problems	F-1

Table 2-1. Start-up Overview (Continued)

What you will need to do ...	Refer to ...	On page ...
Note that the debugger prompt appears.	***Installation Instructions***	2-12
	Debugger General Information.	B-1
	You may also wish to obtain the ***Debugging Package for Motorola 68K CISC CPUs User's Manual*** and the ***177Bug Diagnostics User's Manual,*** listed in ***Related Documentation***	1-9
Initialize the clock.	***Installation Instructions***	2-12
	Debugger General Information	B-1
Examine and/or change environmental parameters.	***Installation Instructions***	2-12
	Environment Command	D-3
Program the PPCchip2 and VMEchip2.	***Memory Maps***	3-4
	You may also wish to obtain the ***Single Board Computers Programmer's Reference Guide,*** listed in ***Related Documentation***	1-9

2

Hardware Preparation

To select the desired configuration and ensure proper operation of the MVME177, certain option modifications may be necessary before installation. The MVME177 provides software control for most of these options. Some options cannot be done in software, so are done by jumpers on headers. Most other modifications are done by setting bits in control registers after the MVME177 has been installed in a system. (The MVME177 registers are described in *Chapter 4*, and/or in the *Single Board Computers Programmer's Reference Guide* as listed in *Related Documentation* in Chapter 1).

The location of switches, jumper headers, connectors, and LED indicators on the MVME177 is illustrated in Figure 2-1.

The MVME177 has been factory tested and is shipped with the factory jumper settings described in the following sections. The MVME177 operates with its required and factory-installed Debug Monitor, MVME177Bug (177Bug), with these factory jumper settings.

Settings can be made for:

- General purpose readable jumpers on header (J1)
- SRAM backup power source select header (J2) (optional)
- System controller header (J6)
- Thermal sensing pins (J7)
- EPROM/Flash configuration jumper (J8)
- Serial port 4 clock configuration select headers (J9 and J10)

Refer to Table 2-2 to configure the jumper settings for each header.

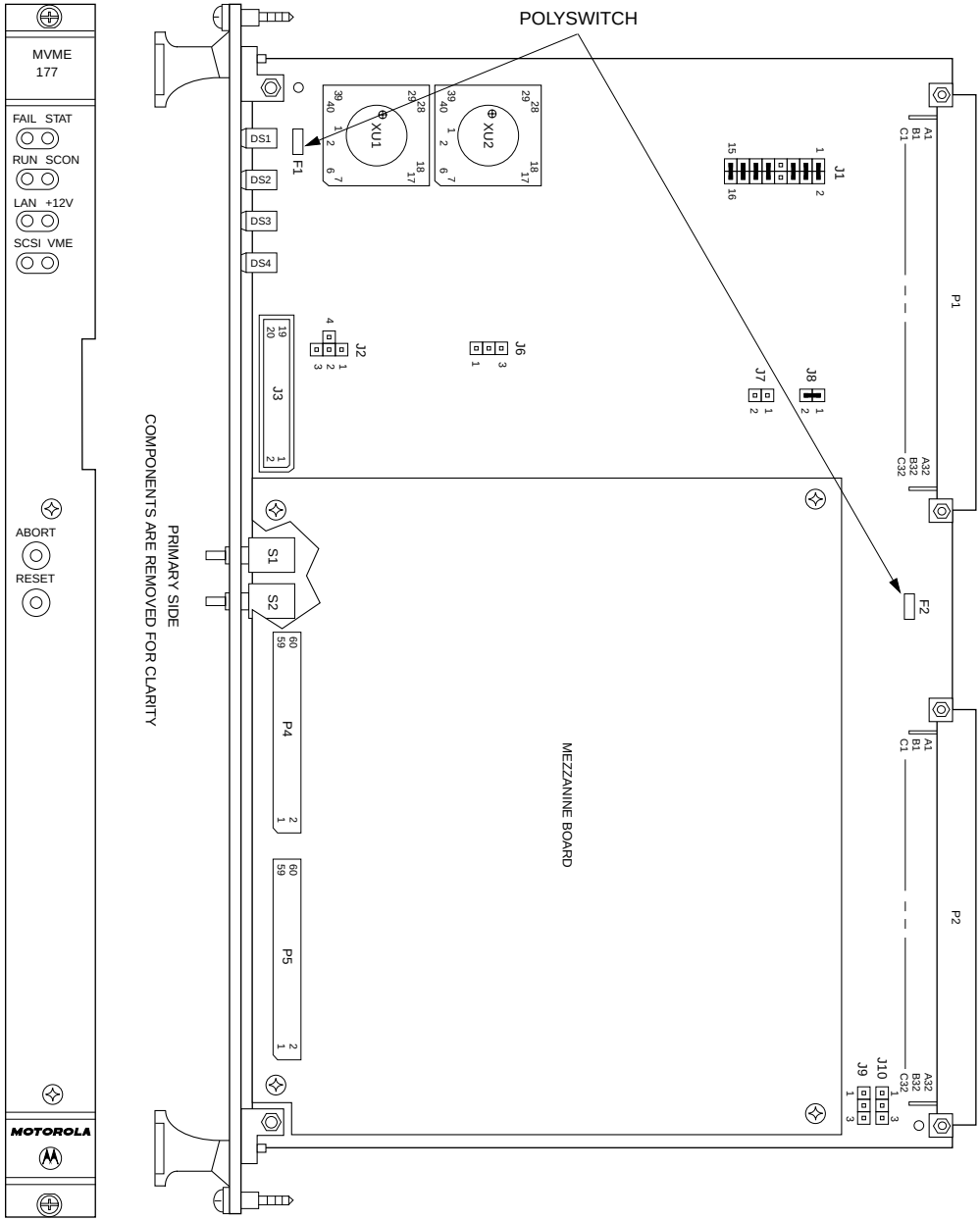


Figure 2-1. MVME177 Switches, Headers, Connectors, Polyswitches, and LEDs

Table 2-2. Configuring MVME177 Headers

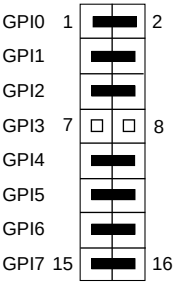
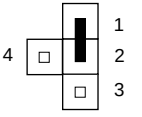
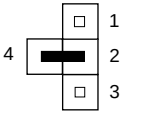
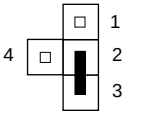
Header Number	Header Description	Configuration	Jumpers		Notes
J1	General purpose software readable jumpers	GPI0 - GPI2: User-definable GPI3: Reserved GPI4 - GPI7: User-definable	1 -- 2 (GPI0) 3 -- 4 (GPI1) 5 -- 6 (GPI2) 9-- 10 (GPI4) 11 -- 12 (GPI5) 13 -- 14 (GPI6) 15 -- 16 (GPI7) (Factory configuration)		1, 2
J2	SRAM backup power source select header	VMEbus +5V STBY	2 -- 1 (Factory configuration)		3
		Backup power disabled	4 -- 2		
		Backup from battery	3 -- 2		

Table 2-2. Configuring MVME177 Headers (Continued)

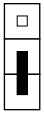
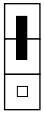
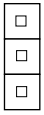
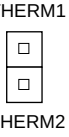
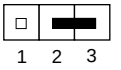
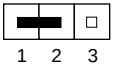
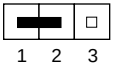
Header Number	Header Description	Configuration	Jumpers		Notes
J6	System controller header	System controller	1 -- 2 (Factory configuration)		4
		Auto system controller	2 -- 3		
		Not system controller	None		
J7	Thermal sensing pins	Connected to MC68060 internal thermal resistor	None (Factory configuration)		5
J8	EPROM/Flash configuration jumper	1MB EPROM and 2MB Flash enabled	1 -- 2 (Factory configuration)		6
		4 MB Flash enabled	None		

Table 2-2. Configuring MVME177 Headers (Continued)

Header Number	Header Description	Configuration	Jumpers		Notes
J9	Serial Port 4 clock configuration select headers	Receive RTXC4	2 -- 3 (Factory configuration)		7
		Drive RTXC4	1 -- 2		
J10		Receive TRXC4	2 -- 3 (Factory configuration)		7
		Drive TRXC4	1 -- 2		

MVME177 Header Notes:	
1.	The general purpose readable jumpers on header J1 can be read as I/O control register 3 (at $FFF40088, bits 0-7) in the VMEchip2 LCSR (see Chapter 4, VMEchip2). The bit values are read as a 1 when the jumper is off, and as a 0 when the jumper is on.
2.	On the MVME177, pins 7 and 8 (bit 3) are removed for board ID and the bit value is reserved.

MVME177 Header Notes: (Continued)	
3.	Header J2 is used to select the power source used to back up the SRAM on the MVME177 when the backup battery is installed. **Caution** — Do not remove all jumpers from J2. This may disable the SRAM. If you remove the battery, you must install jumpers on J2 between pins 2 and 4, as shown for Backup Power Disabled.
4.	The MVME177 can be the VMEbus system controller. The system controller function is enabled, disabled, or configured for automatic select by jumpers on header J6. If set for AUTO SCON, the MVME177 determines if it is the system controller by its position on the bus. If the MVME177 is in the first slot from the left, it configures itself as the system controller. When the MVME177 is system controller, the SCON LED is turned on. The VMEchip2 may be configured as a system controller when J6 is jumpered as shown. **Caution** — Do not jumper J6 to Auto System Controller. At this time, this feature is not functioning properly. Set up jumpers on J6 only as System Controller or Not System Controller. **Note** — AUTO SCON only works with a non-active backplane.
5.	The thermal sensing pins, THERM1 and THERM2, are connected to an internal thermal resistor and provide information about the average temperature of the processor. Refer to the *M68000 Microprocessors User's Manual* for additional information on the use of these pins.
6.	The FLASH jumper, J8, is used to select the Flash memory and EPROM configuration on the MVME177. If the board is configured for 1MB EPROM and 2MB Flash memory, the VMEchip2 GPIO bits can be programmed to select the first or second 2MB of Flash. See Chapter 4 for more information on Flash memory. You can also use the 177Bug **SFLASH** command to map the Flash memory.
7.	Serial port 4 can be configured to use clock signals provided by the RTXC4 and TRXC4 signal lines. Headers J9 and J10 on the MVME177 configure serial port 4 to drive or receive RTXC4 and TRXC4, respectively. Both jumpers should be set the same way. Factory configuration is with port 4 set to receive both signals. The remaining configuration of the clock lines is accomplished using the Serial Port 4 Clock Configuration Select header on the MVME712M transition board. Refer to the *MVME712M Transition Module and P2 Adapter Board User's Manual* for configuration of that header.

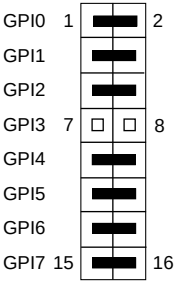
Setup Instructions

Even though the MVME177Bug EPROMs are installed on the MVME177 module in the factory, follow this setup procedure for 177Bug to operate properly with the MVME177.

Caution

Inserting or removing modules while power is applied could damage module components.

1. Turn all equipment power OFF.
2. Refer to Table 2-2 in the *Hardware Preparation* section in this chapter and install/remove jumpers on headers as required for your particular application.
   a. Jumpers on header J1 affect 177Bug operation as listed below. The default condition is with seven jumpers installed between the following pairs of pins:



The MVME177 may be configured with these readable jumpers. These jumpers can be read as a register (at $FFF40088) in the VMEchip2 LCSR. The bit values are read as a one when the jumper is off, and as a zero when the jumper is on. This jumper block (header J1) contains eight bits. Refer to the *Single Board Computers Programmer's Reference Guide*.

The MVME177Bug reserves/defines the four lower order bits (GPI3 to GPI0). The following table shows the bits reserved/defined by the debugger:

Bit	J1 Pins	Description
Bit #0 (GPI0)	1-2	When this bit is a one (high), it instructs the debugger to use local Static RAM for its work page (i.e., variables, stack, vector tables, etc.).
Bit #1 (GPI1)	3-4	When this bit is a one (high), it instructs the debugger to use the default setup/operation parameters in ROM versus the user setup/operation parameters in NVRAM. This is the same as depressing the RESET and ABORT switches at the same time. This feature can be used in the event the user setup is corrupted or does not meet a sanity check. Refer to the **ENV** command (Appendix B) for the ROM defaults.
Bit #2 (GPI2)	5-6	Reserved for future use.
Bit #3 (GPI3)	7-8	Reserved for bug board ID use.
Bit #4 (GPI4)	9-10	Open to your application.
Bit #5 (GPI5)	11-12	Open to your application.
Bit #6 (GPI6)	13-14	Open to your application.
Bit #7 (GPI7)	15-16	Open to your application.

b. Jumpers on headers J2, J6, J7, J8, J9, and J10 configure the board as described in the instructions in Table 2-2.

3. Be sure that the two 256K x 16 177Bug EPROMs are installed in proper sockets on the MVME177 module. Install the odd label (such as B01) EPROM in socket XU1 (for Least Significant Words), and install the even label (such as B02) EPROM in XU2 (for Most Significant Words). Be sure that physical chip orientation is correct, with flatted corner of each EPROM aligned with corresponding portion of EPROM socket on the MVME177 module.

4. This completes the MVME177 Module hardware preparation procedures. Proceed to the next section to install the module in the chassis. Refer to the setup procedure for your particular chassis or system for details concerning the installation of the MVME177.

2

MVME177 Module Installation Instructions

When you have configured the MVME177's headers and installed the selected EPROMs in the sockets as described previously, install the MVME177 module in the system as follows:

1. Turn all equipment power OFF and disconnect the power cable from the AC power source.

Caution

Inserting or removing modules while power is applied could result in damage to module components.

Warning

Dangerous voltages, capable of causing death, are present in this equipment. Use extreme caution when handling, testing, and adjusting.

2. Remove chassis cover as instructed in the equipment user's manual.
3. Remove the filler panel(s) from the appropriate card slot(s) at the front and rear of the chassis (if the chassis has a rear card cage). The MVME177 module requires power from both P1 and P2. It may be installed in any double-height unused card slot, if it is not configured as system controller. If the MVME177 is configured as system controller, it must be installed in the leftmost card slot (slot 1) to correctly initiate the bus-grant daisy-chain and to have proper operation of the IACK-daisy-chain driver. Install the MVME177 in the front of the chassis. You can install the MVME712*x* in the front or the rear of the chassis. Other modules in the system may have to be moved to allow space for the MVME712M which has a double-wide front panel.
4. Carefully slide the MVME177 module into the card slot. Be sure the module is seated properly into the P1 and P2 connectors on the backplane. Do not damage or bend connector pins. Fasten the module in the chassis with screws

provided, making good contact with the transverse mounting rails to minimize RFI emissions.

5. Remove IACK and BG jumpers from the header on the chassis backplane for the card slot in which the MVME177 is installed.

6. Connect the P2 Adapter Board and specified cable(s) to the MVME177 at P2 on the backplane at the MVME177 slot, to mate with (optional) terminals or other peripherals at the EIA-232-D serial ports, parallel port, SCSI ports, and LAN Ethernet port. Refer to the manuals listed in *Related Documentation* in Chapter 1 for information on installing the P2 Adapter Board and the MVME712*x* transition module(s). (Some connection diagrams are in the *Single Board Computers Programmer's Reference Guide*). Some cable(s) are not provided with the MVME712*x* module(s), and therefore are made or provided by the user. (Motorola recommends using shielded cables for all connections to peripherals to minimize radiation). Connect the peripherals to the cable(s). Detailed information on the EIA-232-D signals supported is found in Appendix A.

7. Connect the terminal to be used as the 177Bug system console to the default debug EIA-232-D port at serial port 1 on backplane connector P2 through an MVME712*x* transition module. Refer to the *Single Board Computers Programmer's Reference Guide* for some possible connection diagrams. Set up the terminal as follows:

   - Eight bits per character
   - One stop bit per character
   - Parity disabled (no parity)
   - Baud rate 9600 baud (default baud rate of MVME177 ports at power-up)

   After power-up, the baud rate of the debug port can be reconfigured by using the Port Format (**PF**) command of the 177Bug debugger.

Note In order for high baud-rate serial communication between 177Bug and the terminal to work, the terminal must do some form of handshaking. If the terminal being used does not do hardware handshaking via the CTS line, then it must do XON/XOFF handshaking. If you get garbled messages and missing characters, then you should check the terminal to make sure XON/XOFF handshaking is enabled.

8. If you want to connect devices (such as a host computer system and/or a serial printer) to the other EIA-232-D port connectors (marked SERIAL PORTS 2, 3, and 4 on the MVME712*x* transition module), connect the appropriate cables and configure the port(s) as detailed in step 6 above. After power-up, this(these) port(s) can be reconfigured by programming the MVME177 CD2401 Serial Controller Chip (SCC), or by using the 177Bug **PF** command.

   Note that the MVME177 also contains a parallel port. To use a parallel device, such as a printer, with the MVME177, connect it to the "printer" port at P2 through an MVME712*x* transition module. Refer to the *MVME177 Single Board Computers Programmer's Reference Guide* for some possible connection diagrams. However, you could also use a module such as the MVME335 for a parallel port connection.

9. Install any other required VMEmodules in the system.
10. Replace the chassis cover.
11. Connect power cable to AC power source.
12. Turn equipment power ON. 177Bug executes some self-checks and displays the debugger prompt "`177-Bug>`" (if 177Bug is in Board Mode). However, if the **ENV** command has put 177Bug in System Mode, the system performs a selftest and attempts an autoboot. Refer to the **ENV** and **MENU** commands listed in the Debugger Command Table in Appendix B.

Note that when the MVME177 comes up in a cold reset, 177Bug runs in System Mode. Using the Environment (**ENV**) or **MENU** commands can make 177Bug run in Board Mode. Refer to the Debugger Commands Table in Appendix B.

If the confidence test fails, the test aborts when the first fault is encountered. If possible, an appropriate message displays, and control then returns to the menu.

Refer to Appendix B for general information and operation of the Debugger.

13. At the `177-Bug>` prompt, use the **SET** command to initialize the onboard Real-Time Clock (RTC) and to set the time and date.
14. Use the 177Bug's **ENV** command to verify the NVRAM (BBRAM) parameters, and optionally use **ENV** to make changes to the environmental parameters. Refer to Appendix D for the environment parameters.

System Considerations

The MVME177 draws power from both P1 and P2 of the VMEbus backplane. P2 is also used for the upper 16 bits of data for 32-bit transfers, and for the upper 8 address lines for extended addressing mode. The MVME177 will not operate properly unless its main board is connected to P1 and P2 of the VMEbus backplane.

Whether the MVME177 operates as a VMEbus master or as a VMEbus slave, it is configured for 32 bits of address and for 32 bits of data (A32/D32). However, it handles A16 or A24 devices in the address ranges indicated in *Chapter 3*. D8 and/or D16 devices in the system must be handled by the MC68060 software. Refer to the memory maps in *Chapter 3*.

The MVME177 contains shared onboard DRAM whose base address is software-selectable. Both the onboard processor and offboard VMEbus devices see this local DRAM at base physical address $00000000, as programmed by the MVME177Bug

firmware. This may be changed, by software, to any other base address. Refer to the *Single Board Computers Programmer's Reference Guide* for details.

If the MVME177 attempts to access offboard resources in a nonexistent location, and is not system controller, and if the system does *not* have a global bus time-out, the MVME177 waits forever for the VMEbus cycle to complete. This causes the system to hang up. There is only one situation in which the system might lack this global bus time-out:

- The MVME177 is not the system controller, and
- There is no global bus time-out elsewhere in the system

Multiple MVME177 modules may be configured into a single VME card cage. In general, hardware multiprocessor features are supported.

Other MPUs on the VMEbus can:

- Interrupt
- Disable
- Communicate with, and
- Determine the operational status of

the processor(s). One register of the GCSR set includes four bits which function as location monitors to allow one MVME177 processor to broadcast a signal to other MVME177 processors, if any. All eight registers are accessible from any local processor as well as from the VMEbus.

The MVME177 provides +12 Vdc power to the Ethernet LAN transceiver interface through a 1 amp polyswitch F2 located on the MVME177 module. The +12V LED lights when +12 Vdc is available. The polyswitch is located near diode CR1. Polyswitches act like circuit breakers that reset automatically when the excessive load is removed. If the Ethernet transceiver fails to operate, check

the polyswitch. When using the MVME712M module, the yellow LED (DS1) on the MVME712M front panel lights when LAN power is available, indicating that the polyswitch is good.

The MVME177 provides SCSI terminator power through a diode and a 1 amp polyswitch F1 located on the P2 Adapter Board. If the polyswitch is blown (i.e., open), the SCSI devices may not operate or may function erratically. When the P2 Adapter Board is used with an MVME712M and the SCSI bus is connected to the MVME712M, the green LED (DS2) on the MVME712M front panel lights when there is SCSI terminator power. If the LED flickers during SCSI bus operation, the polyswitch should be checked.

Operating Instructions 3

Introduction

This chapter provides necessary information to use the MVME177 module in a system configuration. This includes:

- Controls and indicators
- Memory maps
- Software initialization of the module

Controls and Indicators

On the front panel of the MVME177 module are the following:

- ABORT and RESET switches
- FAIL, STAT, RUN, SCON, LAN,+12V (LAN power), SCSI, and VME indicators

ABORT Switch S1

When enabled by software, the recessed front panel ABORT switch generates an interrupt at a user-programmable level. It is normally used to abort program execution and return to the 177Bug debugger firmware located in the MVME177 EPROMs.

The ABORT switch interrupter in the VMEchip2 is an edge-sensitive interrupter connected to the ABORT switch. This interrupter is filtered to remove switch bounce.

RESET Switch S2

The recessed front panel RESET switch resets all onboard devices, and drives SYSRESET* if the board is system controller. The RESET switch may be disabled by software.

The VMEchip2 includes both a global and a local reset driver. When the chip operates as the VMEbus system controller, the reset driver provides a global system reset by asserting the VMEbus signal SYSRESET*. A SYSRESET* may be generated by the following:

- RESET switch
- Power up reset
- Watchdog timeout
- Control bit in the LCSR

SYSRESET* remains asserted for at least 200 msec, as required by the VMEbus specification.

Similarly, the VMEchip2 provides an input signal and a control bit to initiate a local reset operation. By setting a control bit, software can maintain a board in a reset state, disabling a faulty board from participating in normal system operation. The local reset driver is enabled even when the VMEchip2 is not the system controller. A local reset may be generated by:

- RESET switch
- Power up reset
- Watchdog timeout
- VMEbus SYSRESET*
- Control bit in the GCSR

Front Panel Indicators (DS1 - DS4)

There are eight LEDs on the MVME177 front panel: FAIL, STAT, RUN, SCON, LAN, +12V (LAN power), SCSI, and VME. The purpose of each LED is as follows:

- The red FAIL LED (part of DS1) lights when the BRDFAIL signal line is active
- The MC68060 status lines are decoded, on the MVME177, to drive the yellow STAT (status) LED (part of DS1). In this case, a halt condition from the processor lights the LED
- The green RUN LED (part of DS2) lights when the local bus TIP* signal line is low. This indicates one of the local bus masters is executing a local bus cycle
- The green SCON LED (part of DS2) lights when the VMEchip2 in the MVME177 is the VMEbus system controller
- The green LAN LED (part of DS3) lights when the LAN chip is local bus master
- The MVME177 supplies +12V power to the Ethernet transceiver interface through a fuse. The green +12V (LAN power) LED (part of DS3) lights when power is available to the transceiver interface
- The green SCSI LED (part of DS4) lights when the SCSI chip is local bus master
- The green VME LED (part of DS4) lights when the board is using the VMEbus (VMEbus AS* is asserted by the VMEchip2) or when the board is accessed by the VMEbus (VMEchip2 is the local bus master)

Memory Maps

There are two possible perspectives or points of view for memory maps:

- The mapping of all resources as viewed by local bus masters (local bus memory map)
- The mapping of onboard resources as viewed by VMEbus Masters (VMEbus memory map)

Local Bus Memory Map

The local bus memory map is split into different address spaces by the transfer type (TT) signals. The local resources respond to the normal access and interrupt acknowledge codes.

There is some address translation capability in the VMEchip2. This allows multiple MVME177s on the same VMEbus with different virtual local bus maps as viewed by different VMEbus masters.

Normal Address Range

The memory map of devices that respond to the normal address range is shown in the following tables. The normal address range is defined by the Transfer Type (TT) signals on the local bus. On the MVME177, Transfer Types 0, 1, and 2 define the normal address range.

Table 3-1. *Local Bus Memory Map*, is the entire map from $00000000 to $FFFFFFFF. Many areas of the map are user-programmable, and suggested uses are shown in the table. The cache inhibit function is programmable in the MMUs. The onboard I/O space must be marked cache inhibit and serialized in its page table.

Table 3-2 on page 3-6 further defines the map for the local I/O devices.

Table 3-1. Local Bus Memory Map

Address Range	Devices Accessed	Port Size	Size	Software Cache Inhibit	Notes
$00000000 - DRAMSIZE	User programmable (onboard ECC DRAM on mezzanine)	D32	DRAMSIZE	N	1, 2
DRAMSIZE - $FF7FFFFF	User programmable (VMEbus A32/A24)	D32/D16	3GB	-	3, 4
$FF800000 - $FFBFFFFF	EPROM/Flash	D32	1MB EPROM/ 4MB Flash	N	1
$FFC00000 - $FFDFFFFF	Reserved	--	2MB	--	5
$FFE00000 - $FFE1FFFF	Onboard SRAM (default)	D32	128KB	N	6
$FFE20000 - $FFEFFFFF	Onboard SRAM (repeated)	D32	896KB	N	6
$FFF00000 - $FFFEFFFF	Local I/O devices (refer to next table)	D32-D8	960KB (1MB-64KB)	Y	3
$FFFF0000 - $FFFFFFFF	User programmable (VMEbus A16)	D32/D16	64KB	-	2, 4

Notes:

1. Flash/EPROM devices appear at $FF800000 through $FFBFFFFF, and also appear at $00000000 through $003FFFFF if the ROM0 bit in the VMEchip2 EPROM control register is high (ROM0 = 1). The ROM0 bit is located at address $FFF40030 bit 20. ROM0 is set to 1 after each reset. The ROM0 bit must be cleared before other resources (DRAM or SRAM) can be mapped in this range ($00000000 through $003FFFFF). The VMEchip2 and DRAM map decoders are disabled by a local bus reset.

   On the MVME177, the Flash/EPROM memory is mapped at $00000000 through $003FFFFF by hardware default through the VMEchip2.

2. This area is user-programmable. The suggested use is shown in the table. The DRAM decoder is programmed in the MCECC chip, and the local-to-VMEbus decoders are programmed in the VMEchip2.

3. Size is approximate.

4. Cache inhibit depends on devices in area mapped.

5. This area is not decoded. If these locations are accessed and the local bus timer is enabled, the cycle times out and is terminated by a TEA signal.

6. The SRAM has optional battery backup on the MVME177.

The following table focuses on the Local I/O Devices portion of the local bus Main Memory Map.

Table 3-2. Local I/O Devices Memory Map

Address Range	Devices Accessed	Port Size	Size	Notes
$FFF00000 - $FFF3FFFF	Reserved	--	256KB	5
$FFF40000 - $FFF400FF	VMEchip2 (LCSR)	D32	256B	1,4
$FFF40100 - $FFF401FF	VMEchip2 (GCSR)	D32-D8	256B	1,4
$FFF40200 - $FFF40FFF	Reserved	--	3.5KB	5,7
$FFF41000 - $FFF41FFF	Reserved	--	4KB	5
$FFF42000 - $FFF42FFF	PCCchip2	D32-D8	4KB	1
$FFF43000 - $FFF430FF	MCECC #1	D8	256B	1
$FFF43100 - $FFF431FF	MCECC #2	D8	256B	1
$FFF43200 - $FFF43FFF	MCECCs (repeated)	--	3.5KB	1,7
$FFF44000 - $FFF44FFF	Reserved	--	4KB	5
$FFF45000 - $FFF451FF	CD2401 (Serial Comm. Cont.)	D16-D8	512B	1,9
$FFF45200 - $FFF45DFF	Reserved	--	3KB	7,9
$FFF45E00 - $FFF45FFF	Reserved	--	512B	1,9
$FFF46000 - $FFF46FFF	82596CA (LAN)	D32	4KB	1,8
$FFF47000 - $FFF47FFF	53C710 (SCSI)	D32/D8	4KB	1
$FFF48000 - $FFF4FFFF	Reserved	--	32KB	5
$FFF50000 - $FFF6FFFF	Reserved	--	128KB	5
$FFF70000 - $FFF76FFF	Reserved	--	28KB	6

Table 3-2. Local I/O Devices Memory Map (Continued)

Address Range	Devices Accessed	Port Size	Size	Notes
$FFF77000 - $FFF77FFF	Reserved	--	4KB	2
$FFF78000 - $FFF7EFFF	Reserved	--	28KB	6
$FFF7F000 - $FFF7FFFF	Reserved	--	4KB	2
$FFF80000 - $FFF9FFFF	Reserved	--	128KB	6
$FFFA0000 - $FFFBFFFF	Reserved	--	128KB	5
$FFFC0000 - $FFFCFFFF	DS1643/MK48T08 (BBRAM, TOD Clock)	D32-D8	64KB	1
$FFFD0000 - $FFFDFFFF	Reserved	--	64KB	5
$FFFE0000 - $FFFEFFFF	Reserved	--	64KB	2

Notes:

1. For a complete description of the register bits, refer to the data sheet for the specific chip. For a more detailed memory map, refer to the following detailed peripheral device memory maps.

2. On the MVME177 this area does not return an acknowledge signal. If the local bus timer is enabled, the access times out and terminates by a TEA signal.

3. Byte reads should be used to read the interrupt vector. These locations do not respond when an interrupt is not pending. If the local bus timer is enabled, the access times out and terminates by a TEA signal.

4. Writes to the LCSR in the VMEchip2 must be 32 bits. LCSR writes of 8 or 16 bits terminate with a TEA signal. Writes to the GCSR may be 8, 16 or 32 bits. Reads to the LCSR and GCSR may be 8, 16 or 32 bits.

5. This area does not return an acknowledge signal. If the local bus timer is enabled, the access times out and terminates by a TEA signal.

6. This area does return an acknowledge signal.

7. Size is approximate.

8. Port commands to the 82596CA must be written as two 16-bit writes: upper word first and lower word second.

9. The CD2401 appears repeatedly from $FFF45200 to $FFF45FFF on the MVME177. If the local bus timer is enabled, the access times out and terminates by a TEA signal.

Software Initialization

Most functions that have been enabled with switches or jumpers on other modules are enabled by setting control registers on the MVME177. At power up or reset, the EPROMs that contain the 177Bug debugging package set up the default values of many of these registers.

Specific programming details may be determined by study of the *M68060 Microprocessor User's Manual*. You can also check the details of all the MVME177 onboard registers as given in the *Single Board Computers Programmer's Reference Guide*.

Multi-MPU Programming Considerations

Good programming practice dictates that only one MPU at a time has control of the MVME177 control registers.

Of particular note are:

- Registers that modify the address map
- Registers that require two cycles to access
- VMEbus interrupt request registers

Local Reset Operation

Local reset (LRST) is a subset of system reset (SRST). Local reset can be generated five ways:

- Expiration of the watchdog timer

- Pressing the front panel RESET switch (if the system controller function is disabled)
- Asserting a bit in the board control register in the GCSR
- SYSRESET*
- Power-up reset

Note The GCSR allows a VMEbus master to reset the local bus. This feature is very dangerous and should be used with caution. The local reset feature is a partial system reset, not a complete system reset such as power-up reset or SYSRESET*. When the local bus reset signal is asserted, a local bus cycle may be aborted. The VMEchip2 is connected to both the local bus and the VMEbus and if the aborted cycle is bound for the VMEbus, erratic operation may result. Communications between the local processor and a VMEbus master should use interrupts or mailbox locations; reset should not be used in normal communications. Reset should be used only when the local processor is halted or the local bus is hung and reset is the last resort.

Any VMEbus access to the MVME177 while it is in the reset state is ignored. If a global bus timer is enabled, a bus error is generated.

Functional Description 4

Introduction

This chapter provides a block diagram level description for the MVME177 module. The functional description provides an overview of the module, followed by a detailed description of several blocks of the module. The block diagram of the MVME177 is shown in Figure 4-1 on page 4-3.

Descriptions of the other blocks of the MVME177, including programmable registers in the ASICs and peripheral chips, are given in the *Single Board Computers Programmer's Reference Guide*. Refer to it for the rest of the functional description of the MVME177 module.

MVME177 Functional Description

The MVME177 is a high functionality VMEbus single board computer designed around the MC68060 chip. The MVME177 has:

- 4/8/16/32/64/128/256MB of dynamic RAM
- SCSI mass storage interface
- Four serial ports
- One parallel port
- Ethernet transceiver interface

Data Bus Structure

The local data bus on the MVME177 is a 32-bit synchronous bus that is based on the MC68060 bus, and supports burst transfers and snooping. The various local bus master and slave devices use the

local bus to communicate. The local bus is arbitrated by priority type arbiter and the priority of the local bus masters from highest to lowest is:

- 82596CA LAN
- CD2401 serial (through the PCCchip2)
- 53C710 SCSI
- VMEbus
- MPU

In the general case, any master can access any slave; however, not all combinations pass the common sense test. Refer to the *Single Board Computers Programmer's Reference Guide* and to the user's guide for each device to determine:

- Port size
- Data bus connection
- Any restrictions that apply when accessing the device

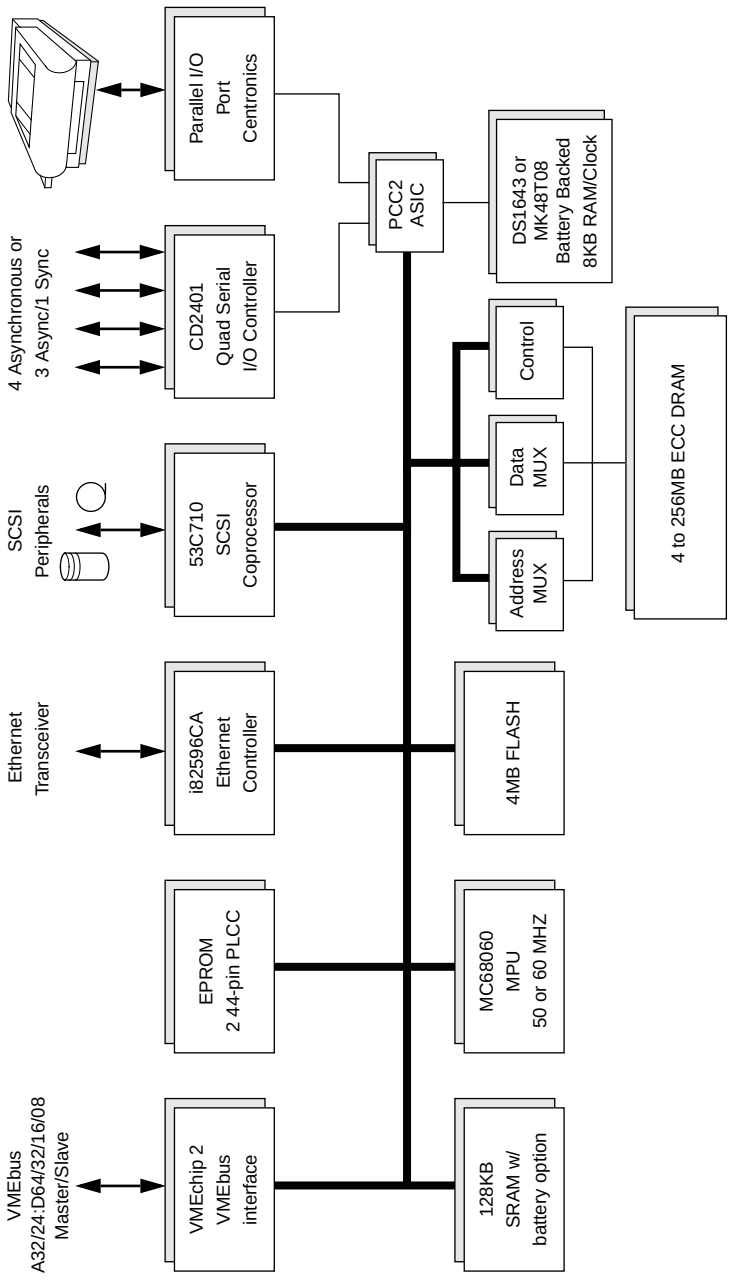


Figure 4-1. MVME177 Block Diagram

MC68060 MPU

The MC68060 microprocessor is the main processor for the MVME177. The superscalar MC68060 processor has:

- Two MC68040-compatible CPU integer cores
- MC68040-compatible floating point core
- Independent 8KB instruction and operand data caches
- MC68040-compatible paged memory management unit
- A bus controller

The processor is in a PGA socket. Its clock speed is 50 MHz (for the -00*x* models), and 60 MHz (for the -01*x* models). Note that the local processor bus runs at only half the processor speed. Refer to the MC68060 user's manual for more information.

Flash Memory and EPROM

Flash Memory

The MVME177 includes four 28F008SA Flash memory devices. The Flash devices provide 4MB of ROM at address $FF800000-$FFBFFFFF. The Flash is organized as one 32-bit bank for 32-bit code execution from the processor. The Flash could, for instance, be used for the onboard debugger firmware (177Bug) which would be downloaded from I/O resources such as:

- Ethernet
- SCSI
- A serial port, or
- The VMEbus

When Flash is used with EPROM, either the top or bottom 2MB of Flash is available in the second 2MB of memory space after the EPROM. Refer to Table 4-1 below.

Table 4-1. EPROM and Flash Control and Configuration

Jumper or Control Bit	Control Condition	Memory Configuration
FLASHJP jumper J8	Jumper in (= low)	2MB EPROM (lower) and 2MB Flash (upper)
	Jumper out (= high)	All 4MB Flash
VMEchip2 bit GPIO2	GPIO2 bit low (and with J8 jumper in)	First 2MB Flash accessible (Note)
	GPIO2 bit high (and with J8 jumper in)	Second 2MB Flash accessible (Note)

Note: These 2MB of Flash will be following the EPROMs in memory if the FLASHJP (J8) jumper is in, and could be read or write depending on the Flash write protect control.

Because only 1M x 8-bit Flash chips are used, there is no user-configured jumper selection block required to pick the Flash chip size.

The memory map for the Flash devices is controlled by the VMEchip2 ASIC. The 32-bit wide Flash can support:

- 8 bit
- 16 bit, and
- 32 bit access

Flash write protection is programmable through the VMEchip2 GPIO register. The address map location of Flash is at $000000 through $3FFFFF at local reset if the FLASHJP jumper (J8) is in, providing for the all-Flash mode. In the mixed EPROM/Flash mode, half of the Flash is accessable at addresses $200000 through $3FFFFF, depending on the condition of the VMEchip2 GPIO2 bit.

Because the MVME177 uses 1M x 8-bit Flash memory devices and EPROMs with no download ROM, the software programs the VMEchip2 ROM0 and REV EROM bits properly so that the Flash/EPROM appears at address $0 after powerup. The hardware is implemented so that the EPROM/Flash appears at address $00000000 following a local bus reset.

The MVME177 implements Flash write protection through clearing a control bit (GPIO1) in the GPIO register in the VMEchip2, to enable write by the software after download process/ programming is completed.

EPROM

There are two 44-pin PLCC/CLCC EPROM sockets for SGS-Thompson M27C4002 (256K x 16) or AMD 27C4096 type EPROMs. They are organized as one 32-bit wide bank that supports:

- 8 bit
- 16 bit, and
- 32-bit read accesses

The EPROMs as shipped are normally used for the onboard debugger firmware (177Bug), but could be used to download user code to Flash. The EPROMs make up only 1MB of memory, but can share the first 2MB of space with the first 2MB of Flash. The EPROMs occupy only 1MB space in the ROM space in mixed mode and will be repeated in the second 1MB space (which is reserved for future expansion). The EPROMs could coexist with this 2MB of Flash, or could be used to program all 4MB of Flash, then the J8 jumper could be removed to make only Flash available.

After a system reset, the EPROMs are mapped to the default addresses $00000 through $FFFFF, and could be mapped to $FF800000 through $FF8FFFFF if needed. The control between mapping EPROM/Flash mixed mode and all Flash mode is done by the combination of external board jumper J8 and the VMEchip2 bit GPIO2. Table 4-1 shows how the “Flash” jumper and GPIO bit 2 change the EPROM/Flash configuration.

The EPROMs/Flashes are mapped to local bus address 0 following a local bus reset. This allows the MC68060 to access the reset vector and execution address following a reset. The EPROMs are controlled by the VMEchip2. The following items are all programmable:

- Map decoder
- Access time
- Time they appear at address 0

For more detail, refer to the VMEchip2 in the *Single Board Computers Programmer's Reference Guide*.

SRAM

The boards include 128KB of 32-bit wide static RAM arrays that are not parity protected and support:

- 8 bit
- 16 bit, and
- 32 bit wide accesses

The SRAM allows the debugger to operate and limited diagnostics to be executed without the DRAM mezzanine. The SRAM will not support burst cycles. The SRAM is controlled by the VMEchip2, and the access time is programmable. Refer to the VMEchip2 in the *Single Board Computers Programmer's Reference Guide* for more detail. The boards are populated with 100 ns SRAMs.

SRAM battery backup is optionally available on the MVME177. The battery backup function is provided by a Dallas DS1210S. Only one backup power source is supported on the MVME177. The battery supplies VCC to the SRAMs when main power is removed.

Each time the MVME177 is powered up, the DS1210S checks the power source. If the voltage of the backup source is less than two volts, the second memory cycle is blocked. This allows software to

provide an early warning to avoid data loss. Because the DS1210S may block the second access, the software should do at least two accesses before relying on the data.

Optionally, the MVME177 provides jumpers that allow the power source of the DS1210S to connect to the VMEbus +5 V STDBY pin or the onboard battery.

The optional power source for the SRAM is a socketed Sanyo CR2430 battery. A small capacitor is provided to allow the battery to be quickly replaced without data loss.

The lifetime of the battery is very dependent on the ambient temperature of the board and the power-on duty cycle. The FB1225 and CR2430 lithium batteries should provide at least two years of backup time with the board powered off and the board at 40° C. If the power-on duty cycle is 50% (the board is powered on half of the time), the battery lifetime is four years. At lower ambient temperatures the backup time is greatly extended and may approach the shelf life of the battery.

When a board is stored, if the battery is present, it should be disconnected to prolong battery life. This is especially important at high ambient temperatures. MVME177 boards with battery backup are shipped with the batteries disconnected.

The power leads from the battery are exposed on the solder side of the board, therefore the board should not be placed on a conductive surface or stored in a conductive bag unless the battery is removed.

Note Lithium batteries incorporate inflammable materials such as lithium and organic solvents. If lithium batteries are mistreated or handled incorrectly, they may burst open and ignite, possibly resulting in injury and/or fire. When dealing with lithium batteries, carefully follow the precautions listed below in order to prevent accidents:

- Do not short circuit

- Do not disassemble, deform, or apply excessive pressure
- Do not heat or incinerate
- Do not apply solder directly
- Do not use different models, or new and old batteries together
- Do not charge
- Always check proper polarity

To remove the battery from the module, carefully pull the battery from the socket.

Onboard DRAM

The MVME177 onboard DRAM is located on a mezzanine board. The mezzanine boards use error checking and correction (ECC) protection to correct single-bit errors and detect double-bit errors. Interrupts or bus exception can be enabled when a bit error is detected. The interrupt output from the memory mezzanine is connected to the VMEchip2 PEIRQ* interrupt input.

Mezzanine board sizes are:

- 4MB
- 8MB
- 16MB
- 32MB
- 64MB
- 128MB (ECC)

Two mezzanine boards may be stacked to provide 256MB of onboard RAM. The main board and a single mezzanine board together take one slot. The stacked configuration requires two VMEboard slots. The DRAM is four-way interleaved to efficiently support cache burst cycles.

The DRAM map decoder can be programmed to accommodate different base address(es) and sizes of mezzanine boards. The onboard DRAM is disabled by a local bus reset and must be programmed before the DRAM can be accessed. Refer to the MCECC in the *Single Board Computers Programmer's Reference Guide* for detailed programming information. Most DRAM devices require some number of access cycles before the DRAMs are fully operational. Normally this requirement is met by the onboard refresh circuitry and normal DRAM initialization. However, software should insure a minimum of 10 initialization cycles are performed to each bank of RAM.

Battery Backed Up RAM and Clock

The DS1643/MK48T08 RAM and clock chip is used on the MVME177. This chip provides the following items, all in one 28-pin package:

- Time of day clock
- Oscillator
- Crystal
- Power fail detection
- Memory write protection
- 8KB of RAM
- A battery

The clock provides:

- Seconds

- Minutes
- Hours
- Day
- Date
- Month
- Year

in BCD 24-hour format. Corrections for 28-, 29- (leap year), and 30-day months are automatically made. No interrupts are generated by the clock. The DS1643/MK48T08 is an 8 bit device; however, the interface provided by the PCCchip2 supports:

- 8 bit
- 16 bit, and
- 32 bit accesses to the DS1643/MK48T08

Refer to the PCCchip2 in the *Single Board Computers Programmer's Reference Guide* and to the DS1643/MK48T08 data sheet for detailed programming information.

VMEbus Interface

The local bus to VMEbus interface, the VMEbus to local bus interface, and the local-VMEbus DMA controller functions on the MVME177 are provided by the VMEchip2. The VMEchip2 can also provide the VMEbus system controller functions. Refer to the VMEchip2 in the *Single Board Computers Programmer's Reference Guide* for detailed programming information.

I/O Interfaces

The MVME177 provides onboard I/O for many system applications. The I/O functions include:

- Serial ports

- Parallel (printer) port
- Ethernet transceiver interface
- SCSI mass storage interface

Serial Port Interface

The CD2401 serial controller chip (SCC) is used to implement the four serial ports. The serial ports support the standard baud rates (110 to 38.4K baud). The four serial ports are different functionally because of the limited number of pins on the P2 I/O connector. Serial port 1 is a minimum function asynchronous port. It uses:

- RXD
- CTS
- TXD
- RTS

Serial ports 2 and 3 are full function asynchronous ports. They use:

- RXD
- CTS
- DCD
- TXD
- RTS
- DTR

Serial port 4 is a full function asynchronous or synchronous port. It can operate at synchronous bit rates up to 64 k bits per second. It uses:

- RXD
- CTS
- DCD

- TXD
- RTS
- DTR

It also interfaces to the synchronous clock signal lines. Refer to the *Single Board Computers Programmer's Reference Guide* for drawings of the serial port interface connections.

All four serial ports use EIA-232-D drivers and receivers located on the main board, and all the signal lines are routed to the I/O connector. The configuration headers are located on the main board and the MVME712*x* transition board. An external I/O transition board such as the MVME712*x* should be used to convert the I/O connector pinout to industry-standard connectors.

Note The MVME177 board hardware ties the DTR signal from the CD2401 to the pin labeled RTS at connector P2. Likewise, RTS from the CD2401 is tied to DTR on P2. Therefore, when programming the CD2401, assert DTR when you want RTS, and RTS when you want DTR.

The interface provided by the PCCchip2 allows the 16-bit CD2401 to appear at contiguous addresses; however, accesses to the CD2401 must be 8 or 16 bits. 32-bit accesses are not permitted. Refer to the CD2401 data sheet and to the PCCchip2 in the *Single Board Computers Programmer's Reference Guide* for detailed programming information.

The CD2401 supports DMA operations to local memory. Because the CD2401 does not support a retry operation necessary to break VMEbus lockup conditions, the CD2401 DMA controllers should not be programmed to access the VMEbus. The hardware does not restrict the CD2401 to onboard DRAM.

Parallel Port Interface

The PCCchip2 provides an 8-bit bidirectional parallel port. All eight bits of the port must be either inputs or outputs (no individual selection). In addition to the 8 bits of data, there are two control pins and five status pins. Each of the status pins can generate an interrupt to the MPU in any of the following programmable conditions:

- High level
- Low level
- High-to-low transition
- Low-to-high transition

This port may be used as a Centronics- compatible parallel printer port or as a general parallel I/O port.

When used as a parallel printer port, the five status pins function as Printer:

- Acknowledge (ACK)
- Fault (FAULT*)
- Busy (BSY)
- Select (SELECT)
- Paper Error (PE)

The control pins act as:

- Printer Strobe (STROBE*)
- Input Prime (INP*)

The PCCchip2 provides an auto-strobe feature similar to that of the MVME147 PCC. In auto-strobe mode, after a write to the Printer Data Register, the PCCchip2 automatically asserts the STROBE* pin for a selected time specified by the Printer Fast Strobe control bit. In manual mode, the Printer Strobe control bit directly controls the state of the STROBE* pin.

Refer to the *Single Board Computers Programmer's Reference Guide* for drawings of the printer port interface connections.

Ethernet Interface

The 82596CA is used to implement the Ethernet transceiver interface. The 82596CA accesses local RAM using DMA operations to perform its normal functions. Because the 82596CA has small internal buffers and the VMEbus has an undefined latency period, buffer overrun may occur if the DMA is programmed to access the VMEbus. Therefore, the 82596CA should not be programmed to access the VMEbus.

Every MVME177 is assigned an Ethernet Station Address. The address is $08003E2*xxxxx*, where *xxxxx* is the unique 5-nibble number assigned to the board (i.e., every MVME177 has a different value for *xxxxx*).

Each module has an Ethernet Station Address displayed on a label attached to the VMEbus P2 connector. In addition, the six bytes including the Ethernet address are stored in the configuration area of the BBRAM. That is, 08003E2*xxxxx* is stored in the BBRAM. At an address of $FFFC1F2C, the upper four bytes (08003E2*x*) can be read. At an address of $FFFC1F30, the lower two bytes (*xxxx*) can be read. Refer to the BBRAM, TOD Clock memory map description in *Chapter 3*. The MVME177 debugger has the capability to retrieve or set the Ethernet address.

If the data in the BBRAM is lost, the user should use the number on the VMEbus P2 connector label to restore it.

The Ethernet transceiver interface is located on the MVME177 main module, and the industry standard connector is located on the MVME712*x* transition module.

Support functions for the 82596CA are provided by the PCCchip2. Refer to the 82596CA user's guide and to the *Single Board Computers Programmer's Reference Guide* for detailed programming information.

SCSI Interface

The MVME177 provides for mass storage subsystems through the industry-standard SCSI bus. These subsystems may include:

- Hard and floppy disk drives
- Streaming tape drives
- Other mass storage devices

The SCSI interface is implemented using the NCR 53C710 SCSI I/O controller.

The SCSI clock input to the 53C710 is fixed at 50 MHz, in order to have higher SCSI bus performance, and to make it easier for software to program the 53C710 controller when the MC68060 processor speed changes.

Support functions for the 53C710 are provided by the PCCchip2. Refer to the 53C710 user's guide and to the *Single Board Computers Programmer's Reference Guide* for detailed programming information.

SCSI Termination

The system configurer must ensure that the SCSI bus is properly terminated at both ends. On the MVME177, sockets are provided for the terminators on the P2 transition board. If the SCSI bus ends at the P2 transition board, then termination resistors must be installed on the P2 transition board. +5V power to the SCSI bus TERM power line and termination resistors is provided through a fuse on the MVME712 transition board, and a diode located on the P2 transition board.

Local Resources

The MVME177 includes many resources for the local processor. These include:

- Tick timers
- Software programmable hardware interrupts
- Watchdog timer

- Local bus time-out

Note The time basis for all local resources is set by Prescaler register(s). Refer to the *Single Board Computers Programmer's Reference Guide* for detailed programming information.

Programmable Tick Timers

Four 32-bit programmable tick timers with 1 µs resolution are provided:

- Two in the VMEchip2 and
- Two in the PCCchip2

The tick timers can be programmed to generate periodic interrupts to the processor. Refer to the VMEchip2 and PCCchip2 in the *Single Board Computers Programmer's Reference Guide* for detailed programming information.

Watchdog Timer

A watchdog timer function is provided in the VMEchip2. When the watchdog timer is enabled, it must be reset by software within the programmed time or it times out. The watchdog timer can be programmed to generate:

- A SYSRESET signal
- Local reset signal, or
- Board fail signal if it times out

Refer to the VMEchip2 in the *Single Board Computers Programmer's Reference Guide* for detailed programming information.

Software-Programmable Hardware Interrupts

Eight software-programmable hardware interrupts are provided by the VMEchip2. These interrupts allow software to create a hardware interrupt. Refer to the VMEchip2 in the *Single Board Computers Programmer's Reference Guide* for detailed programming information.

Local Bus Time-out

The MVME177 provides a time-out function for the local bus. When the timer is enabled and a local bus access times out, a Transfer Error Acknowledge (TEA) signal is sent to the local bus master. The time-out value is selectable by software for:

- 8 µsec
- 64 µsec
- 256 µsec
- Infinite

The local bus timer does not operate during VMEbus bound cycles. VMEbus bound cycles are timed by the VMEbus access timer and the VMEbus global timer. Refer to the VMEchip2 in the *Single Board Computers Programmer's Reference Guide* for detailed programming information.

Module Identification

Software distinguishes between an MVME177 module and an MVME176 module by use of the I/O control register (GPI) bit 3. On an MVME177, the I/O control register (GPI) bit 3 is out (open) for a "high" (one). On an MVME176, the I/O control register (GPI) bit 3 is hardwired in (shorted) for a "low" (zero).

Timing Performance

This section provides the performance information for the MVME177. Various MVME177s are designed to operate at 50 MHz or 60 MHz (when supported by 060).

Local Bus to DRAM Cycle Times

The PCCchip2 and VMEchip2 have the same local bus interface timing as the MC68060, therefore the following cycle times also apply to the PCCchip2 and the VMEchip2. Read accesses to

onboard DRAM require 5 bus clock cycles with the bus error reported in the current cycle. Write accesses to onboard DRAM require 2 bus clock cycles.

Burst read accesses require 8 (5-1-1-1) bus clock cycles with the bus error reported in the current cycle. Burst write cycles require 5 (2-1-1-1) bus clock cycles.

ROM Cycle Times

The ROM cycle time is programmable from 4 to 11 bus clock cycles. The data transfers are 32 bits wide. Refer to the *Single Board Computers Programmer's Reference Guide*.

SCSI Transfers

The MVME177 includes a SCSI mass storage bus interface with DMA controller. The SCSI DMA controller uses a FIFO buffer to interface the 8-bit SCSI bus to the 32-bit local bus. The FIFO buffer allows the SCSI DMA controller to efficiently transfer data to the local bus in four longword bursts. This reduces local bus usage by the SCSI device.

The first longword transfer of a burst, with snooping disabled, requires:

- Four bus clocks with parity off, and
- Five bus clocks with parity on

Each of the remaining three transfers requires one bus clock.

The transfer rate of the DMA controller is 44MB/sec at 25 MHz with parity off. Assuming a continuous transfer rate of 5MB/sec on the SCSI bus, 12% of the local bus bandwidth is used by transfers from the SCSI bus.

Note The actual SCSI bus transfer rate is fixed, no matter what the speed of the microprocessor.

LAN DMA Transfers

The MVME177 includes a LAN interface with DMA controller. The LAN DMA controller uses a FIFO buffer to interface the serial LAN bus to the 32-bit local bus. The FIFO buffer allows the LAN DMA controller to efficiently transfer data to the local bus.

The 82596CA does not execute MC68060 compatible burst cycles, therefore the LAN DMA controller does not use burst transfers. DRAM write cycles require 3 clock cycles, and read cycles require:

- 5 clock cycles with parity off and
- 6 clock cycles with parity on

The transfer rate of the LAN DMA controller is 20MB/sec at 25 MHz (or 24MB/sec at 30 MHz) with parity off. Assuming a continuous transfer rate of 1MB/sec on the LAN bus, 5% (or 4%) of the local bus bandwidth is used by transfers from the LAN bus.

Remote Status and Control

The remote status and control connector, J3, is a 20-pin connector located behind the front panel of the MVME177. It provides system designers the flexibility to access critical indicator and reset functions. This allows a system designer to construct a RESET/ABORT/LED panel that can be located remotely from the MVME177.

In addition to the LED and the RESET and ABORT switches access, this connector also includes:

- Two general purpose TTL-level I/O pins
- One general purpose interrupt pin which can also function as a trigger input. This interrupt pin is level programmable

EIA-232-D Interconnections

A

Introduction

The EIA-232-D standard is the most widely used terminal/computer and terminal/modem interface, and yet it is not fully understood. This may be because not all the lines are clearly defined, and many users do not see the need to follow the standard in their applications. Often designers think only of their own equipment, but the state of the art is computer-to-computer or computer-to-modem operation. A system should easily connect to any other system.

The EIA-232-D standard was originally developed by the Bell System to connect terminals via modems. Several handshaking lines were included for that purpose. Although handshaking is unnecessary in many applications, the lines themselves remain part of many designs because they facilitate troubleshooting.

Table A-1 lists the standard EIA-232-D interconnections. To interpret this information correctly, remember that EIA-232-D was intended to connect a terminal to a modem. When computers are connected to each other without modems, one of them must be configured as a terminal (data terminal equipment: DTE) and the other as a modem (data circuit-terminating equipment: DCE). Since computers are normally configured to work with terminals, they are said to be configured as a modem in most cases.

Signal levels must lie between +3 and +15 volts for a high level, and between -3 and -15 volts for a low level. Connecting units in parallel may produce out-of-range voltages and is contrary to EIA-232-D specifications.

Table A-1. EIA-232-D Interconnections

Pin Number	Signal Mnemonic	Signal Name and Description
01		Not used.
02	TxD	TRANSMIT DATA. Data to be transmitted; input to the modem from the terminal.
03	RxD	RECEIVE DATA. Data which is demodulated from the receive line; output from the modem to the terminal.
04	RTS	REQUEST TO SEND. Input to the modem from the terminal when required to transmit a message. With RTS off, the modem carrier remains off. When RTS is turned on, the modem immediately turns on the carrier.
05	CTS	CLEAR TO SEND. Output from the modem to the terminal to indicate that message transmission can begin. When a modem is used, CTS follows the off-to-on transition of RTS after a time delay.
06	DSR	DATA SET READY. Output from the modem to the terminal to indicate that the modem is ready to transmit data.
07	SIG-GND	SIGNAL GROUND. Common return line for all signals at the modem interface.
08	DCD	DATA CARRIER DETECT. Output from the modem to the terminal to indicate that a valid carrier is being received.
09-14		Not used.
15	TxC	TRANSMIT CLOCK (DCE). Output from the modem to the terminal; clocks data from the terminal to the modem.
16		Not used.
17	RxC	RECEIVE CLOCK. Output from the modem to the terminal; clocks data from the modem to the terminal.
18, 19		Not used.
20	DTR	DATA TERMINAL READY. Input to the modem from the terminal; indicates that the terminal is ready to send or receive data.
21		Not used.

Table A-1. EIA-232-D Interconnections (Continued)

Pin Number	Signal Mnemonic	Signal Name and Description
22	RI	RING INDICATOR. Output from the modem to the terminal; indicates to the terminal that an incoming call is present. The terminal causes the modem to answer the phone by carrying DTR true while RI is active.
23		Not used.
24	TxC	TRANSMIT CLOCK (DTE). Input to modem from terminal; same function as TxC on pin 15.
25	BSY	BUSY. Input to modem from terminal. A positive EIA signal applied to this pin causes the modem to go off-hook and make the associated phone busy.

Notes:

1. A high EIA-232-D signal level is +3 to +15 volts. A low level is -3 to -15 volts. Connecting units in parallel may produce out-of-range voltages and is contrary to specifications.

2. The EIA-232-D interface is intended to connect a terminal to a modem. When computers are connected without modems, one must be configured as a modem and the other as a terminal.

Levels of Implementation

There are several levels of conformance that may be appropriate for typical EIA-232-D interconnections. The bare minimum requirement is the two data lines and a ground. The full implementation of EIA-232-D requires 12 lines; it accommodates:

- Automatic dialing
- Automatic answering
- Synchronous transmission

A middle-of-the-road approach is illustrated in Figure A-1.

Signal Adaptations

One set of handshaking signals frequently implemented are RTS and CTS. CTS is used in many systems to inhibit transmission until the signal is high. In the modem application, RTS is turned around and returned as CTS after 150 microseconds. RTS is programmable in some systems to work with the older type 202 modem (half duplex). CTS is used in some systems to provide flow control to avoid buffer overflow. This is not possible if modems are used. It is usually necessary to make CTS high by connecting it to RTS or to some source of +12 volts such as the resistors shown in Figure A-1. CTS is also frequently jumpered to an MC1488 gate which has its inputs grounded (the gate is provided for this purpose).

Another signal used in many systems is DCD. The original purpose of this signal was to inform the system that the carrier tone from the distant modem was being received. This signal is frequently used by the software to display a message like `CARRIER NOT PRESENT` to help the user to diagnose failure to communicate. Obviously, if the system is designed properly to use this signal and is not connected to a modem, the signal must be provided by a pullup resistor or gate as described above (see Figure A-1).

Many modems expect a DTR high signal and issue a DSR response. These signals are used by software to help prompt the operator about possible causes of trouble. The DTR signal is sometimes used to disconnect the phone circuit in preparation for another automatic call. These signals are necessary in order to communicate with all possible modems (see Figure A-1).

Sample Configurations

Figure A-1 is a good minimum configuration that almost always works. If the CTS and DCD signals are not received from the modem, the jumpers can be moved to artificially provide the needed signal.

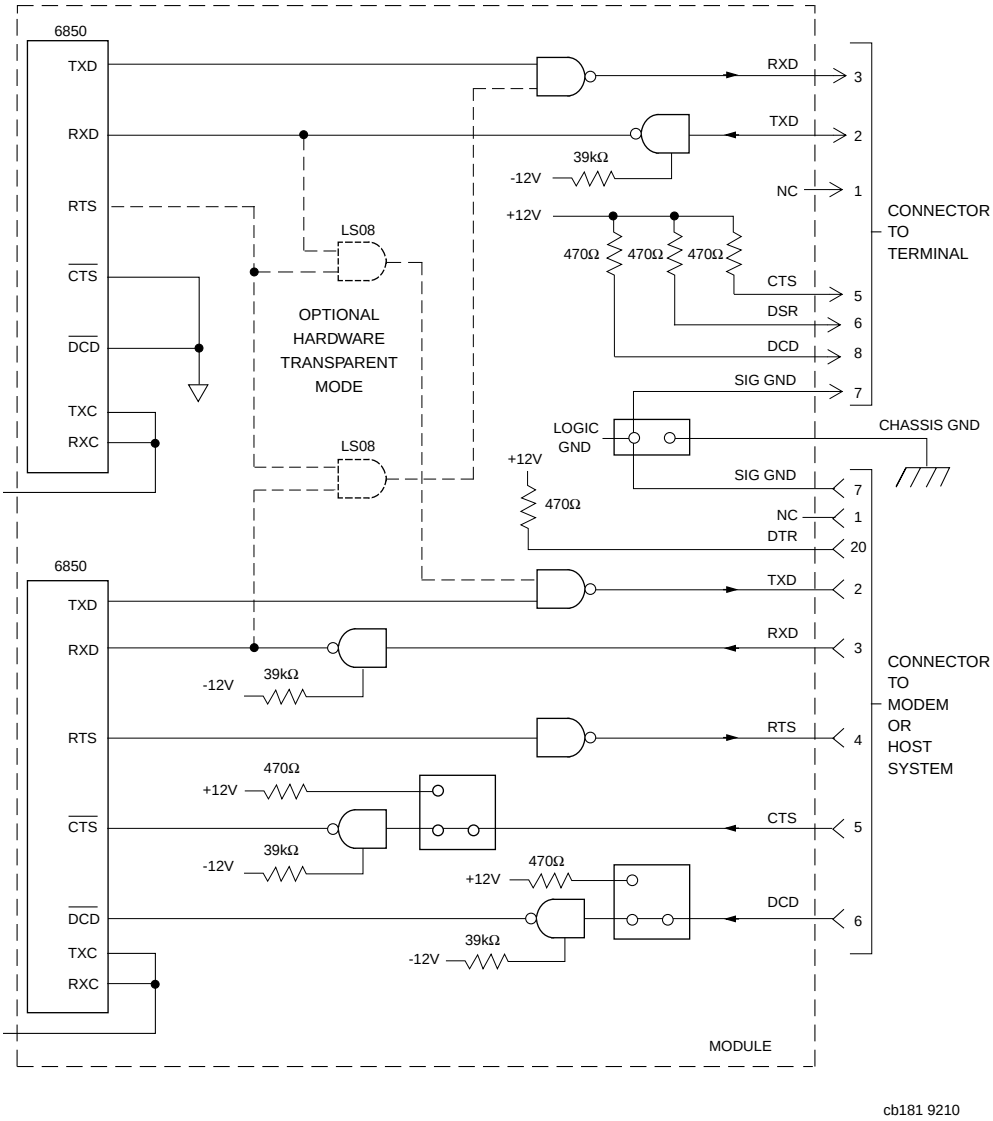


Figure A-1. Middle-of-the-Road EIA-232-D Configuration

Figure A-2 shows a way of wiring an EIA-232-D connector to enable a computer to connect to a basic terminal with only three lines. This is feasible because most terminals have a DTR signal that is ON, and which can be used to pull up the CTS, DCD, and DSR signals.

Two of these connectors wired back-to-back can be used. In this implementation, however, diagnostic messages that might otherwise be generated do not occur because all the handshaking is bypassed. In addition, the TX and RX lines may have to be crossed since TX from a terminal is outgoing but the TX line on a modem is an incoming signal.

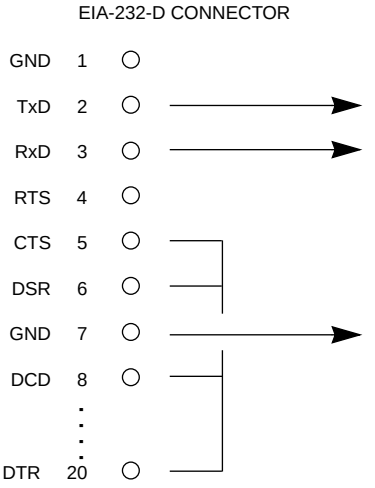


Figure A-2. Minimum EIA-232-D Connection

Proper Grounding

Another subject to consider is the use of ground pins. There are two pins labeled GND. Pin 7 is the SIGNAL GROUND and must be connected to the distant device to complete the circuit. Pin 1 is the CHASSIS GROUND, but it must be used with care. The chassis is connected to the power ground through the green wire in the power cord and must be connected to the chassis to be in compliance with the electrical code.

The problem is that when units are connected to different electrical outlets, there may be several volts of difference in ground potential. If pin 1 of each device is interconnected with the others via cable, several amperes of current could result. This condition may not only be dangerous for the small wires in a typical cable, but may also produce electrical noise that causes errors in data transmission. That is why Figure A-1 shows no connection for pin 1. Normally, pin 7 should only be connected to the CHASSIS GROUND at one point; if several terminals are used with one computer, the logical place for that point is at the computer. The terminals should not have a connection between the logic ground return and the chassis.

Debugger General Information

B

Overview of M68000 Firmware

The firmware for the M68000-based (68K) series of board and system level products has a common genealogy, deriving from the debugger firmware currently used on all Motorola M68000-based CPU modules. The M68000 firmware family provides:

- A high degree of functionality
- User friendliness
- Portability
- Ease of maintenance

This member of the M68000 firmware family is implemented on the MVME177 Single Board Computer, and is known as the MVME177Bug, or simply 177Bug.

Description of 177Bug

The 177Bug package is a powerful evaluation and debugging tool for systems built around the MVME177 CISC-based microcomputers. Facilities are available for loading and executing user programs under complete operator control for system evaluation. 177Bug includes:

- Commands for display and modification of memory
- Breakpoint and tracing capabilities
- A powerful assembler/disassembler useful for patching programs
- A self-test at power-up feature which verifies the integrity of the system

- Various 177Bug routines that handle I/O, data conversion, and string functions available to user programs through the TRAP #15 system calls

177Bug consists of three parts:

- A command-driven user-interactive software debugger, described in this appendix, and hereafter referred to as "the debugger" or "177Bug"
- A command-driven diagnostic package for the MVME177 hardware, hereafter referred to as "the diagnostics"
- A user interface which accepts commands from the system console terminal

When using 177Bug, you operate out of either the debugger directory or the diagnostic directory. If you are in the debugger directory, the debugger prompt "177-Bug>" displays and you have all of the debugger commands at your disposal. If you are in the diagnostic directory, the diagnostic prompt "177-Diag>" displays and you have all of the diagnostic commands at your disposal as well as all of the debugger commands. You may switch between directories by using the Switch Directories (**SD**) command, or may examine the commands in the particular directory that you are currently in by using the Help (**HE**) command.

Because 177Bug is command-driven, it performs its various operations in response to user commands entered at the keyboard. When you enter a command, 177Bug executes the command and the prompt reappears. However, if you enter a command that causes execution of user target code (e.g., **"GO"**), then control may or may not return to 177Bug, depending on the outcome of the user program.

If you have used one or more of Motorola's other debugging packages, you will find the CISC 177Bug very similar. Considerable effort has also been made to make the interactive commands more consistent. For example, delimiters between commands and arguments may now be commas or spaces interchangeably.

177Bug Implementation

MVME177Bug is written largely in the "C" programming language, providing benefits of portability and maintainability. Where necessary, assembler has been used in the form of separately compiled modules containing only assembler code - no mixed language modules are used.

Physically, 177Bug is contained in two 44-pin PLCC/CLCC EPROMs, providing 512KB (128K longwords) of storage. Both EPROMs are necessary regardless of how much space is actually occupied by the firmware, because of the 32-bit longword-oriented MC68060 memory bus architecture. The executable code is checksummed at every power-on or reset firmware entry, and the result (which includes a pre-calculated checksum contained in the EPROMs) is tested for an expected zero. Thus, users are cautioned against modification of the EPROMs unless re-checksum precautions are taken. The power-on defaults for the MVME177 debug port are:

- Eight bits per character
- One stop bit per character
- Parity disabled (no parity)
- Baud rate 9600 baud (default baud rate of MVME177 ports at power-up)

After power-up, the baud rate of the debug port can be reconfigured by using the Port Format (**PF**) command of the 177Bug debugger.

Autoboot

Autoboot is a software routine that is contained in the several 177Bug EPROMs to provide an independent mechanism for booting an operating system. This autoboot routine automatically scans for controllers and devices in a specified sequence until a valid bootable device containing a boot media is found, or the list is

exhausted. If a valid bootable device is found, a boot from that device begins. The controller scanning sequence goes from the lowest controller Logical Unit Number (LUN) detected to the highest LUN detected.

At power-up, Autoboot is enabled, and providing the drive and controller numbers encountered are valid, the following message displays on the system console:

```
"Autoboot in progress... To abort hit <BREAK>"
```

Following this message there is a delay to allow you an opportunity to abort the Autoboot process if you wish. Then the actual I/O begins: the program pointed to within the volume ID of the media specified loads into RAM and control passes to it. If, however, during this time you want to gain control without Autoboot, you can press the:

- BREAK key
- Software ABORT switch
- RESET switch

Autoboot is controlled by parameters contained in the **ENV** command. These parameters allow:

- Selection of specific boot devices
- Selection of files
- Programming of the Boot delay

Refer to the **ENV** command in the Commands Table for more details.

Caution

Although streaming tape can be used to autoboot, the same power supply must be connected to the:

- Streaming tape drive
- Controller
- MVME177

At power-up, the tape controller positions the streaming tape to load point where the volume ID can correctly be read and used.

If, however, the MVME177 loses power but the controller does not, and the tape happens to be at load point, the sequences of commands required (attach and rewind) cannot be given to the controller and autoboot will not be successful.

ROMboot

The ROMboot function is configured/enabled by the Environment (**ENV**) command (refer to Commands Table at the end of this Appendix) and executes:

- At power-up
- At reset (optionally)
- By the **RB** command, assuming there is valid code in the EPROMs (or optionally elsewhere on the module or VMEbus) to support it.

If ROMboot code is installed, a user-written routine is given control (if the routine meets the format requirements). One use of ROMboot might be resetting SYSFAIL* on an unintelligent controller module. The **NORB** command disables the function.

For a user's ROMboot module to gain control through the ROMboot linkage, four requirements must be met:

- Power must have just been applied (but the **ENV** command can change this to also respond to any reset)
- Your routine must be located within the MVME177 ROM memory map (but the **ENV** command can change this to any other portion of the onboard memory, or even offboard VMEbus memory)

- The ASCII string “BOOT” must be located within the specified memory range
- Your routine must pass a checksum test, which ensures that this routine was really intended to receive control at power-up

For complete details on how to use ROMboot, refer to the *Debugging Package for Motorola 68K CISC CPUs User’s Manual*.

Network Boot

Network Auto Boot is a software routine contained in the 177Bug EPROMs that provides a mechanism for booting an operating system using a network (local Ethernet interface) as the boot device. The Network Auto Boot routine automatically scans for controllers and devices in a specified sequence until a valid bootable device containing a boot media is found or the list is exhausted. If a valid bootable device is found, a boot from that device begins. The controller scanning sequence goes from the lowest controller Logical Unit Number (LUN) detected to the highest LUN detected.

At power-up, Network Boot is enabled, and providing the drive and controller numbers encountered are valid, the following message displays on the system console:

```
"Network Boot in progress... To abort hit <BREAK>"
```

Following this message there is a delay to allow you to abort the Auto Boot process if you wish. Then the actual I/O begins: the program pointed to within the volume ID of the media specified loads into RAM and control passes to it. If, however, during this time you want to gain control without Network Boot, you can pres the:

- BREAK key
- Software ABORT switch
- RESET switch

Network Auto Boot is controlled by parameters contained in the **NIOT** and **ENV** commands. These parameters allow:

- Selection of specific boot devices
- Selection of systems
- Selection of files
- Programming of the Boot delay

Refer to the **ENV** and **NIOT** commands in the Commands Table in this Appendix for more details. Also refer to the **ENV** parameters in Appendix D.

Restarting the System

You can initialize the system to a known state in three different ways:

- Reset
- Abort
- Break

Each has characteristics which make it more appropriate than the others in certain situations.

The debugger has a special feature upon a reset condition. This feature is activated by depressing the RESET and ABORT switches at the same time. This feature instructs the debugger to use the default setup/operation parameters in ROM versus your setup/operation parameters in NVRAM. This feature can be used in the event your setup/operation parameters are corrupted or fail to pass a sanity check.

Reset

Pressing and releasing the MVME177 front panel RESET switch initiates a system reset. COLD and WARM reset modes are available. By default, 177Bug is in COLD mode. During COLD reset, a total system initialization occurs, as if the MVME177 had just been powered up. In other words, during a COLD reset:

- All static variables (including disk device and controller parameters) restore to their default states
- The breakpoint table and offset registers are cleared
- The target registers are invalidated
- Input and output character queues are cleared
- Onboard devices (timer, serial ports, etc.) are reset
- The *first* two serial ports are reconfigured to their default state

During WARM reset, the 177Bug preserves the following:

- Variables
- Tables
- Target state registers
- Breakpoints

Reset must be used if the processor ever halts, or if the 177Bug environment is ever lost (vector table is destroyed, stack corrupted, etc.).

Abort

You can initiate an abort by pressing and releasing the ABORT switch on the MVME177 front panel. If an abort is initiated while executing a user program (running target code), a "snapshot" of the processor state is captured and stored in the target registers. For this reason, abort is most appropriate when terminating a user

program that is being debugged. Abort should be used to regain control if the program gets caught in a loop, etc. The target PC and register contents assist you in locating the malfunction.

Pressing and releasing the ABORT switch generates a local board condition that causes:

- A processor interrupt (if enabled)
- The target registers (reflecting the machine state at the time the ABORT switch was pressed) display on the screen
- All breakpoints installed in your code are removed
- Breakpoint table remains intact
- Control returns to the debugger

Break

You can generate a "Break" by pressing and releasing the BREAK key on the terminal keyboard. Break does not generate an interrupt. The only time break is recognized is when characters are sent or received by the console port. A Break causes:

- All breakpoints in your code to be removed
- Breakpoint table to be maintained intact
- A snapshot to be taken of the machine state if the function was entered using SYSCALL
- The snapshot is accessible to you for diagnostic purposes

Often it is desirable to terminate a debugger command prior to its completion; for example, during the display of a large block of memory. Break allows you to terminate the command immediately.

SYSFAIL* Assertion/Negation

Upon a reset/power-up condition the debugger asserts the VMEbus SYSFAIL* line (refer to the VMEbus specification). SYSFAIL* stays asserted if any of the following has occurred:

- Confidence test failure
- NVRAM checksum error
- NVRAM low battery condition
- Local memory configuration status
- Self test (if system mode) has completed with error
- MPU clock speed calculation failure

After debugger initialization is done and none of the above situations have occurred, the SYSFAIL* line is negated. This indicates to the user or VMEbus masters the state of the debugger. In a multi-computer configuration, other VMEbus masters could view the pertinent control and status registers to determine which CPU is asserting SYSFAIL*. SYSFAIL* assertion/negation is also affected by the **ENV** command.

MPU Clock Speed Calculation

The clock speed of the microprocessor is calculated and checked against a user definable parameter housed in NVRAM (refer to the **CNFG** command in the Commands Table). If the check fails, a warning message displays. The calculated clock speed is also checked against known clock speeds and tolerances.

Memory Requirements

The program portion of 177Bug is approximately 512KB of code, consisting of:

- Download
- Debugger
- Diagnostic packages

and is contained entirely in EPROM. The EPROM sockets on the MVME177 are mapped starting at location $FF800000.

177Bug requires a minimum of 64KB of contiguous read/write memory to operate.

The **ENV** command controls where this block of memory is located. Regardless of where the onboard RAM is located, the first 64KB is used for 177Bug stack and static variable space and the rest is reserved as user space. Whenever the MVME177 is reset:

- Target PC is initialized to the address corresponding to the beginning of the user space
- Target stack pointers are initialized to addresses within the user space
- Target Interrupt Stack Pointer (ISP) set to the top of the user space

At power-up or reset, all 8KB of memory at addresses $FFE0C000 through $FFE0DFFF is completely changed by the 177Bug initial stack.

B

Terminal Input/Output Control

When entering a command at the prompt, the following control codes may be entered for limited command line editing.

Note The presence of the caret (^) before a character indicates that the Control (CTRL) key must be held down while striking the character key.

^X	(cancel line)	The cursor is backspaced to the beginning of the line. If the terminal port is configured with the hardcopy or TTY option (refer to PF command), then a carriage return and line feed is issued along with another prompt.
^H	(backspace)	The cursor is moved back one position. The character at the new cursor position is erased. If the hardcopy option is selected, a "/" character is typed along with the deleted character.
^D	(redisplay)	The entire command line as entered so far is redisplayed on the following line.
^A	(repeat)	Repeats the previous line. This happens only at the command line. The last line entered is redisplayed but not executed. The cursor is positioned at the end of the line. You may enter the line as is or you can add more characters to it. You can edit the line by backspacing and typing over old characters.
<DEL>	(delete or rubout)	Performs the same function as **^H**.

When observing output from any 177Bug command, the XON and XOFF characters which are in effect for the terminal port may be entered to control the output, if the XON/XOFF protocol is enabled

(default). These characters are initialized to **^S** and **^Q** respectively by 177Bug, but you may change them with the **PF** command. In the initialized (default) mode, operation is as follows:

^S	(wait)	Console output is halted.
^Q	(resume)	Console output is resumed.

Disk I/O Support

177Bug can initiate disk input/output by communicating with intelligent disk controller modules over the VMEbus. Disk support facilities built into 177Bug consist of:

- Command-level disk operations
- Disk I/O system calls (only via one of the TRAP #15 instructions) for use by user programs
- Defined data structures for disk parameters

Parameters such as:

- Address where the module is mapped
- Device type
- Number of devices attached to the controller module

are kept in tables by 177Bug. Default values for these parameters are assigned at power-up and cold-start reset, but may be altered as described in the section on default parameters, later in this chapter.

Blocks Versus Sectors

The logical block defines the unit of information for disk devices. A disk is viewed by 177Bug as a storage area divided into logical blocks. By default, the logical block size is set to 256 bytes for every block device in the system. You can change the block size on a per device basis with the **IOT** command.

The sector defines the unit of information for the media itself, as viewed by the controller. The sector size varies for different controllers, and the value for a specific device can be displayed and changed with the **IOT** command.

When a disk transfer is requested:

- Start and size of the transfer is specified in blocks
- 177Bug translates this into an equivalent sector specification
- Passes the sector specification on to the controller to initiate the transfer

If the conversion from blocks to sectors yields a fractional sector count, an error is returned and no data is transferred.

Device Probe Function

A device probe with entry into the device descriptor table is performed whenever a specified device is accessed; i.e., when system calls:

- .DSKRD
- .DSKWR
- .DSKCFIG
- .DSKFMT
- .DSKCTRL

or debugger commands:

- BH
- BO
- IOC
- IOP
- IOT
- MAR
- MAW

are used.

The device probe mechanism utilizes the SCSI commands "Inquiry" and "Mode Sense". If the specified controller is non-SCSI, the probe simply returns a status of "device present and unknown". The device probe makes an entry into the device descriptor table with the pertinent data. After an entry has been made, the next time a probe is done it simply returns with "device present" status (pointer to the device descriptor).

Disk I/O via 177Bug Commands

The 177Bug commands listed in the following paragraphs are provided for disk I/O. Detailed instructions for their use are found in the *Debugging Package for Motorola 68K CISC CPUs User's Manual*. When a command is issued to a particular controller LUN and device LUN, these LUNs are remembered by 177Bug so that the next disk command defaults to use the same controller and device.

IOI (Input/Output Inquiry)

This command probes the system for all possible CLUN/DLUN combinations and displays inquiry data for devices which support it. The device descriptor table has space for a maximum of 16 device descriptors. Use the **IOI** command to view the table or clear it if necessary.

IOP (Physical I/O to Disk)

IOP allows you to:

- Read blocks of data
- Write blocks of data
- Format a specified device in a certain way

IOP creates a command packet from the arguments you have specified, then invokes the proper system call function to carry out the operation.

IOT (I/O Teach)

IOT allows you to change any configurable parameters and attributes of the device. In addition, it allows you to view the controllers available in the system.

IOC (I/O Control)

IOC allows you to send command packets as defined by the particular controller directly. **IOC** can also be used to examine the resultant device packet after using the **IOP** command.

BO (Bootstrap Operating System)

BO reads an operating system or control program from the specified device into memory, then transfers control to it.

BH (Bootstrap and Halt)

BH reads an operating system or control program from a specified device into memory, then returns control to 177Bug. It is used as a debugging tool.

Disk I/O via 177Bug System Calls

All operations that actually access the disk are done directly or indirectly by 177Bug TRAP #15 system calls. (The command-level disk operations provide a convenient way of using these system calls without writing and executing a program).

The following system calls allow user programs to perform disk I/O:

.DSKRD	Disk read. Use this system call to read blocks from a disk into memory.
.DSKWR	Disk write. Use this system call to write blocks from memory onto a disk.

.DSKCFIG	Disk configure. Use this system call to change the configuration of the specified device.
.DSKFMT	Disk format. Use this system call to send a format command to the specified device.
.DSKCTRL	Disk control. Use this system call to implement any special device control functions that cannot be accommodated easily with any of the other disk functions.

Refer to the *Debugging Package for Motorola 68K CISC CPUs User's Manual* for information on using these and other system calls.

To perform a disk operation, 177Bug must eventually present a particular disk controller module with a controller command packet which has been especially prepared for that type of controller module. (This is accomplished in the respective controller driver module.) A command packet for one type of controller module usually does not have the same format as a command packet for a different type of module. The system call facilities which perform disk I/O:

- Accept a generalized (controller-independent) packet format as an argument
- Translate it into a controller-specific packet
- Send it to the specified device

Refer to the system call descriptions in the *Debugging Package for Motorola 68K CISC CPUs User's Manual* for details on the format and construction of these standardized "user" packets.

The packets which a controller module expects to receive vary from controller to controller. The disk driver module for the particular hardware module (board) must take the standardized packet given to a trap function and create a new packet which is specifically tailored for the disk drive controller receiving it. Refer to documentation on the particular controller module for the format of its packets, and for using the **IOC** command.

Default 177Bug Controller and Device Parameters

177Bug initializes the parameter tables for a default configuration of controllers and devices (refer to Appendix C). If the system needs to be configured differently than this default configuration (for example, to use a 70MB Winchester drive where the default is a 40MB Winchester drive), then these tables must be changed.

There are three ways to change parameter table contents:

- Using **BO** or **BH**. When you invoke one of these commands, the configuration area of the disk is read and the parameters corresponding to that device are rewritten according to the parameter information contained in the configuration area. This is a temporary change. If a cold-start reset occurs, then the default parameter information is written back into the tables.
- Using the **IOT**. You can use this command to reconfigure the parameter table manually for any controller and/or device that is different from the default. This is also a temporary change and is overwritten if a cold-start reset occurs.
- Obtain the source. You can then change the configuration files and rebuild 177Bug using different defaults. Changes made to the defaults are permanent until changed again.

Disk I/O Error Codes

177Bug returns an error code if an attempted disk operation is unsuccessful.

Network I/O Support

The Network Boot Firmware provides the capability to boot the CPU through the ROM debugger using a network (local Ethernet interface) as the boot device.

The booting process executes in two distinct phases:

- The first phase: the diskless remote node discovers its network identify and the name of the file to be booted
- The second phase: the diskless remote node reads the boot file across the network into its memory

The various modules (capabilities) and the dependencies of these modules that support the overall network boot function are described in the following paragraphs

Intel 82596 LAN Coprocessor Ethernet Driver

This driver manages/surrounds the Intel 82596 LAN Coprocessor. Management is in the scope of:

- Reception of packets
- Transmission of packets
- Receive buffer flushing
- Interface initialization

This module ensures that the packaging and unpackaging of Ethernet packets is performed correctly in the Boot PROM.

UDP/IP Protocol Modules

The Internet Protocol (IP) is designed for use in interconnected systems of packet-switched computer communication networks. The Internet protocol provides for transmitting of blocks of data called datagrams (hence User Datagram Protocol, or UDP) from sources to destinations, where sources and destinations are hosts identified by fixed length addresses.

The UDP/IP protocols are necessary for the TFTP and BOOTP protocols; TFTP and BOOTP require a UDP/IP connection.

RARP/ARP Protocol Modules

The Reverse Address Resolution Protocol (RARP) basically consists of:

- An identity-less node broadcasting a “whoami” packet onto the Ethernet
- The node awaiting an answer
- The RARP server filling an Ethernet reply packet with the target's Internet Address
- The RARP server sending it to the node

The Address Resolution Protocol (ARP) basically provides a method of converting protocol addresses (e.g., IP addresses) to local area network addresses (e.g., Ethernet addresses). The RARP protocol module supports systems which do not support the BOOTP protocol (next paragraph).

BOOTP Protocol Module

The Bootstrap Protocol (BOOTP) basically allows a diskless client machine to discover:

- Its own IP address
- The address of a server host
- The name of a file to be loaded into memory and executed

TFTP Protocol Module

The Trivial File Transfer Protocol (TFTP) is a simple protocol to transfer files. It is implemented on top of the Internet User Datagram Protocol (UDP or Datagram) so it may be used to move files between machines on different networks implementing UDP. The only thing it can do is read and write files from/to a remote server.

Network Boot Control Module

The “control” capability of the Network Boot Control Module is needed to tie together all the necessary modules (capabilities) and to sequence the booting process. The booting sequence consists of two phases:

- “Address determination and bootfile selection” phase
- “File transfer” phase

The first phase utilizes the RARP/BOOTP capability and the second phase utilizes the TFTP capability.

Network I/O Error Codes

177Bug returns an error code whenever an attempted network operation is unsuccessful.

Multiprocessor Support

The MVME177 dual-port RAM feature makes the shared RAM available to remote processors as well as to the local processor. This can be done by either of the following two methods. Either method can be enabled/disabled by the **ENV** command as its Remote Start Switch Method.

Multiprocessor Control Register (MPCR) Method

A remote processor can initiate program execution in the local MVME177 dual-port RAM by issuing a remote **GO** command using the Multiprocessor Control Register (MPCR). The MPCR, located at shared RAM location of $800 offset from the base address the debugger loads it at, contains one of two longwords used to control communication between processors. The MPCR contents are organized as follows:

$800	*	N/A	N/A	N/A	(MPCR)

The status codes stored in the MPCR are of two types:

- Status returned (from the monitor)
- Status set (by the bus master)

The status codes that may be returned from the monitor are:

HEX	0	(HEX 00)	--	Wait. Initialization not yet complete.
ASCII	R	(HEX 52)	--	Ready. The firmware monitor is watching for a change.
ASCII	E	(HEX 45)	--	Code pointed to by the MPAR address is executing.

The status codes that may be set by the bus master are:

ASCII	G	(HEX 47)	--	Use Go Direct (**GD**) logic specifying the MPAR address.
ASCII	B	(HEX 42)	--	Install breakpoints using the Go (**G**) logic.

The Multiprocessor Address Register (MPAR), located in shared RAM location of $804 offset from the base address the debugger loads it at, contains the second of two longwords used to control communication between processors. The MPAR contents specify the address at which execution for the remote processor is to begin if the MPCR contains a G or B. The MPAR is organized as follows:

$804	*	*	*	*	(MPAR)

At power-up, the debug monitor self-test routines initialize RAM, including the memory locations used for multi-processor support ($800 through $807).

The MPCR contains $00 at power-up, indicating that initialization is not yet complete. As the initialization proceeds, the execution path comes to the “prompt” routine. Before sending the prompt, this routine places an R in the MPCR to indicate that initialization is complete. Then the prompt is sent.

If no terminal is connected to the port, the MPCR is still polled to determine whether an external processor requires control to be passed to the dual-port RAM. If a terminal does respond, the MPCR is polled for the same purpose while the serial port is being polled for user input.

An ASCII G placed in the MPCR by a remote processor indicates that the Go Direct type of transfer is requested. An ASCII B in the MPCR indicates that breakpoints are to be armed before control is transferred (as with the **GO** command).

In either sequence, an E is placed in the MPCR to indicate that execution is underway just before control is passed to RAM. (Any remote processor could examine the MPCR contents).

If the code being executed in dual-port RAM is to reenter the debug monitor, a TRAP #15 call using function $0063 (SYSCALL .RETURN) returns control to the monitor with a new display prompt. Note that each time the debug monitor returns to the prompt, an R is moved into the MPCR to indicate that control can be transferred once again to a specified RAM location.

GCSR Method

A remote processor can initiate program execution in the local MVME177 dual-port RAM by issuing a remote **GO** command using the VMEchip2 Global Control and Status Registers (GCSR). The remote **GO** command causes the following sequence:

- Remote processor places the MVME177 execution address in general purpose registers 0 and 1 (GPCSR0 and GPCSR1)
- Remote processor sets bit 8 (SIG0) of the VMEchip2 LM/SIG register
- MVME177 installs breakpoints and begins execution

The result is identical to the MPCR method (with status code B) described in the previous section.

The GCSR registers are accessed in the VMEbus short I/O space. Each general purpose register is two bytes wide, occurring at an even address. The general purpose register number 0 is at an offset of $8 (local bus) or $4 (VMEbus) from the start of the GCSR registers. The local bus base address for the GCSR is $FFF40100. The VMEbus base address for the GCSR depends on the group select value and the board select value programmed in the Local Control

and Status Registers (LCSR) of the MVME177. The execution address is formed by reading the GCSR general purpose registers in the following manner:

GPCSR0	used as the upper 16 bits of the address
GPCSR1	used as the lower 16 bits of the address

The address appears as:

GPCSR0	GPCSR1

Diagnostic Facilities

The 177Bug hardware diagnostics are intended for testing and troubleshooting of the MVME177.

In order to use the diagnostics, you must switch to the diagnostic directory. You may switch between directories by using the **SD** (Switch Directories) command. You may view a list of the commands in the directory that you are currently in by using the **HE** (Help) command.

If you are in the debugger directory, the debugger prompt `177-Bug>` displays, and all of the debugger commands are available. Diagnostics commands cannot be entered at the `177-Bug>` prompt.

If you are in the diagnostic directory, the diagnostic prompt: `177-Diag>` displays, and all of the debugger and diagnostic commands are available.

The diagnostic test groups are listed in the following table. Refer to the *177Bug Diagnostics User's Manual* for complete descriptions of the diagnostic routines available in each test group and instructions on how to invoke them.

Table B-1. Diagnostic Test Groups

Test Group	Description
RAM	Local RAM Tests
SRAM	Static RAM Tests
RTC	MK48T0x Real-Time Clock Tests
PCC2	Peripheral Channel Controller Tests
MCECC	Memory Board Tests
MEMC1	MC040 Memory Controller 1 ASIC Tests
MEMC2	MC040 Memory Controller 2 ASIC Tests
ST2401	CD2401 Serial Port Tests
CMMU	Cache and Memory Management Unit Tests
VME2	VME Interface ASIC VMEchip2 Tests
LANC	LAN Coprocessor (Intel 82596) Tests
NCR	NCR 53C710 SCSI I/O Processor Tests
FLASH	Flash Memory Tests

Notes 1. You may enter command names in either uppercase or lowercase.

2. Some diagnostics depend on restart defaults that are set up only in a particular restart mode. Refer to the documentation on a particular diagnostic for the correct mode.

Using the 177Bug Debugger

Entering Debugger Command Lines

177Bug is command-driven and performs its various operations in response to user commands entered at the keyboard. When the debugger prompt (`177-Bug>`) appears on the terminal screen, then the debugger is ready to accept commands.

As the command line is entered, it is stored in an internal buffer. Execution begins only after the carriage return is entered, so that you can correct entry errors, if necessary, using the control characters described in Chapter 3.

When you enter a command, the debugger executes the command and the prompt reappears. However, if the command entered causes execution of user target code, for example **GO**, then control may or may not return to the debugger, depending on what the user program does. For example, if a breakpoint has been specified, then control returns to the debugger when the breakpoint is encountered during execution of the user program. Alternately, the user program could return to the debugger by means of the TRAP #15 function ".RETURN".

In general, a debugger command is made up of the following parts:

a. The command identifier (i.e., **MD** or **md** for the Memory Display command). Note that either upper- or lowercase is allowed.

b. A port number if the command is set up to work with more than one port.

c. At least one intervening space before the first argument.

d. Any required arguments, as specified by command.

e. An option field, set off by a semicolon (**;**) to specify conditions other than the default conditions of the command.

The commands are shown using a modified Backus-Naur form syntax. The metasymbols used are:

boldface strings	A boldface string is a literal such as a command or a program name, and is to be typed just as it appears.
italic strings	An italic string is a "syntactic variable" and is to be replaced by one of a class of items it represents.
|	A vertical bar separating two or more items indicates that a choice is to be made; only one of the items separated by this symbol should be selected.
[ ]	Square brackets enclose an item that is optional. The item may appear zero or one time.
{ }	Braces enclose an optional symbol that may occur zero or more times.

Syntactic Variables

The following syntactic variables are encountered in the command descriptions which follow. In addition, other syntactic variables may be used and are defined in the particular command description in which they occur.

DEL	Delimiter; either a comma or a space.
EXP	Expression (described in detail in a following section).
ADDR	Address (described in detail in a following section).
COUNT	Count; the syntax is the same as for *EXP*.
RANGE	A range of memory addresses which may be specified either by *ADDR DEL ADDR* or by *ADDR*: *COUNT*.
TEXT	An ASCII string of up to 255 characters, delimited at each end by the single quote mark (').

Expression as a Parameter

An expression can be one or more numeric values separated by the arithmetic operators:

- Plus (+)
- Minus (-)
- Multiplied by (*)
- Divided by (/)
- Logical AND (&)
- Shift left (<<)
- Shift right (>>)

Numeric values may be expressed in either:

- Hexadecimal
- Decimal
- Octal
- Binary

by immediately preceding them with the proper base identifier.

Data Type	Base	Identifier	Examples
Integer	Hexadecimal	$	$FFFFFFFF
Integer	Decimal	&	&1974, &10-&4
Integer	Octal	@	@456
Integer	Binary	%	%1000110

If no base identifier is specified, then the numeric value is assumed to be hexadecimal.

A numeric value may also be expressed as a string literal of up to four characters. The string literal must begin and end with the single quote mark ('). The numeric value is interpreted as the concatenation of the ASCII values of the characters. This value is right-justified, as any other numeric value would be.

String Literal	Numeric Value (In Hexadecimal)
'A'	41
'ABC'	414243
'TEST'	54455354

Evaluation of an expression is performed according to the following rules:

- Always from left to right unless parentheses are used to group part of the expression
- There is no operator precedence
- Subexpressions within parentheses are evaluated first
- Nested parenthetical subexpressions are evaluated from the inside out

Valid expression examples:

Expression	Result (In Hex)	Notes
FF0011	FF0011	
45+99	DE	
&45+&99	90	
@35+@67+@10	5C	
%10011110+%1001	A7	
88<<4	880	shift left
AA&F0	A0	logical AND

The total value of the expression must be between 0 and $FFFFFFFF.

Address as a Parameter

Many commands use *ADDR* as a parameter. The syntax accepted by 177Bug is similar to the one accepted by the MC68060 one-line assembler. All control addressing modes are allowed. An "address + offset register" mode is also provided.

Address Formats

Table B-2 summarizes the address formats which are acceptable for address parameters in debugger command lines.

Table B-2. Debugger Address Parameter Formats

Format	Example	Description
N	140	Absolute address+contents of automatic offset register.
N+R*n*	130+R5	Absolute address+contents of the specified offset register (not an assembler-accepted syntax).
(A*n*)	(A1)	Address register indirect (also post-increment, predecrement)
(*d*,A*n*) or *d*(A*n*)	(120,A1) 120(A1)	Address register indirect with displacement (two formats accepted).
(*d*,A*n*,*Xn*) or *d*(A*n*,*Xn*)	(&120,A1,D2) &120(A1,D2)	Address register indirect with index and displacement (two formats accepted).
([*bd*,A*n*,*Xn*],*od*)	([C,A2,A3],&100)	Memory indirect preindexed.
([*bd*,A*n*],*Xn*,*od*)	([12,A3],D2,&10)	Memory indirect postindexed.
For the memory indirect modes, fields can be omitted. For example, three of many permutations are as follows:		
([,A*n*],*od*)	([,A1],4)	
([*bd*])	([FC1E])	
([*bd*,,*Xn*])	([8,,D2])	

Notes *N* — Absolute address(any valid expression).

A*n* — Address register *n*.
Xn — Index register *n* (A*n* or D*n*).
d — Displacement (any valid expression).
bd — Base displacement (any valid expression).
od — Outer displacement (any valid expression).
n — Register number (0 to 7).
R*n* — Offset register *n*.

Note In commands with *RANGE* specified as *ADDR DEL ADDR*, and with size option W or L chosen, data at the second (ending) address is acted on only if the second address is a proper boundary for a word or longword, respectively

Offset Registers

Eight pseudo-registers (R0 through R7) called offset registers are used to simplify the debugging of relocatable and position-independent modules. The listing files in these types of programs usually start at an address (normally 0) that is not the one at which they are loaded, so it is harder to correlate addresses in the listing with addresses in the loaded program. The offset registers solve this problem by taking into account this difference and forcing the display of addresses in a relative address+offset format. Offset registers have adjustable ranges and may even have overlapping ranges. The range for each offset register is set by two addresses:

- Base
- Top

Specifying the base and top addresses for an offset register sets its range. In the event that an address falls in two or more offset registers' ranges, the one that yields the least offset is chosen.

Note Relative addresses are limited to 1MB (5 digits), regardless of the range of the closest offset register.

Example:

A portion of the listing file of an assembled, relocatable module is shown below:

```
    1
    2                                   *
    3                                   * MOVE STRING SUBROUTINE
    4                                   *
    5   0 00000000 48E78080            MOVESTR  MOVEM.L  D0/A0,-(A7)
    6   0 00000004 4280                          CLR.L    D0
    7   0 00000006 1018                          MOVE.B   (A0)+,D0
    8   0 00000008 5340                          SUBQ.W   #1,D0
    9   0 0000000A 12D8                LOOP      MOVE.B   (A0)+,(A1)+
   10   0 0000000C 51C8FFFC            MOVS      DBRA     D0,LOOP
   11   0 00000010 4CDF0101                      MOVEM.L  (A7)+,D0/A0
   12   0 00000014 4E75                          RTS
   13
   14                                           END
****** TOTAL ERRORS     0--
****** TOTAL WARNINGS   0--
```

The above program was loaded at address $0001327C.

The disassembled code is shown next:

```
177Bug>MD 1327C;DI
0001327C 48E78080                   MOVEM.L  D0/A0,-(A7)
00013280 4280                       CLR.L    D0
00013282 1018                       MOVE.B   (A0)+,D0
00013284 5340                       SUBQ.W   #1,D0
00013286 12D8                       MOVE.B   (A0)+,(A1)+
00013288 51C8FFFC                   DBF      D0,$13286
0001328C 4CDF0101                   MOVEM.L  (A7)+,D0/A0
00013290 4E75                       RTS
177Bug>
```

By using one of the offset registers, the disassembled code addresses can be made to match the listing file addresses as follows:

```
177Bug>OF R0
R0 =00000000 00000000? 1327C. <CR>
177Bug>MD 0+R0;DI <CR>
00000+R0 48E78080                MOVEM.L  D0/A0,–(A7)
00004+R0 4280                    CLR.L    D0
00006+R0 1018                    MOVE.B   (A0)+,D0
00008+R0 5340                    SUBQ.W   #1,D0
0000A+R0 12D8                    MOVE.B   (A0)+,(A1)+
0000C+R0 51C8FFFC                DBF      D0,$A+R0
00010+R0 4CDF0101                MOVEM.L  (A7)+,D0/A0
00014+R0 4E75                    RTS
177Bug>
```

For additional information about the offset registers, refer to the *Debugging Package for Motorola 68K CISC CPUs User's Manual*.

Port Numbers

Some 177Bug commands allow you the option of choosing the port to be used to input or output. Valid port numbers which may be used for these commands are as follows:

1. MVME177 EIA-232-D Debug (Terminal Port 0 or 00) (PORT 1 on the MVME177 P2 connector). Sometimes known as the "console port", it is used for interactive user input/output by default.
2. MVME177 EIA-232-D (Terminal Port 1 or 01) (PORT 2 on the MVME177 P2 connector). Sometimes known as the "host port", this is the default for:
   - Downloading
   - Uploading
   - Concurrent mode
   - Transparent modes.

Note These logical port numbers (0 and 1) are shown in the pinouts of the MVME177 module as "SERIAL PORT 1" and "SERIAL PORT 2", respectively. Physically, they are all part of connector P2.

Entering and Debugging Programs

There are various ways to enter a user program into system memory for execution. One way is to create the program using the Memory Modify (**MM**) command with the assembler/disassembler option. You enter the program one source line at a time. After each source line is entered, it is assembled and the object code loads into memory. Refer to the *Debugging Package for Motorola 68K CISC CPUs User's Manual* for complete details of the 177Bug Assembler / Disassembler.

Another way to enter a program is to download an object file from a host system. The program must be in S-record format (described in the *Debugging Package for Motorola 68K CISC CPUs User's Manual*) and may have been assembled or compiled on the host system. Alternately, the program may have been previously created using the 177Bug **MM** command as outlined above and stored to the host using the Dump (**DU**) command. A communication link must exist between the host system and the MVME177 port 1. (Hardware configuration details are in the section on *Installation and Start-Up* in Chapter 3.) The file is downloaded from the host to MVME177 memory by the Load (**LO**) command.

Another way is by reading in the program from disk, using one of the following disk commands:

- **BO**
- **BH**
- **IOP**

Once the object code has been loaded into memory, you can:

- Set breakpoints
- Run the code
- Trace through the code

B

Calling System Utilities from User Programs

A convenient way of doing character input/output and many other useful operations has been provided so that you do not have to write these routines into the target code. You can access various 177Bug routines via one of the MC68060 TRAP instructions, using vector #15. Refer to the *Debugging Package for Motorola 68K CISC CPUs User's Manual* for details on the various TRAP #15 utilities available and how to invoke them from within a user program.

Preserving the Debugger Operating Environment

This section explains how to avoid contaminating the operating environment of the debugger. 177Bug uses certain of the MVME177 onboard resources and also offboard system memory to contain temporary variables, exception vectors, etc. If you disturb resources upon which 177Bug depends, then the debugger may function unreliably or not at all.

If your application enables translation through the Memory Management Units (MMUs), and if your application utilizes resources of the debugger (e.g., system calls), your application must create the necessary translation tables for the debugger to have access to its various resources.The debugger honors the enabling of the MMUs; it does not disable translation.

177Bug Vector Table and Workspace

As described in the *Memory Requirements* section in Chapter 3, 177Bug needs 64KB of read/write memory to operate. The 177Bug reserves a 1024-byte area for a user program vector table area and then allocates another 1024-byte area and builds an exception vector table for the debugger itself to use. Next, 177Bug:

- Reserves space for static variables
- Initializes these static variables to predefined default values

- Allocates space for the system stack
- Initializes the system stack pointer to the top of this area

With the exception of the first 1024-byte vector table area, you must be extremely careful not to use the above-mentioned memory areas for other purposes. You should refer to the *Memory Requirements* section in Chapter 3 to determine how to dictate the location of the reserved memory areas. If, for example, your program inadvertently wrote over the static variable area containing the serial communication parameters, these parameters would be lost, resulting in a loss of communication with the system console terminal. If your program corrupts the system stack, then an incorrect value may be loaded into the processor Program Counter (PC), causing a system crash.

Hardware Functions

The only hardware resources used by the debugger are the EIA-232-D ports, which are initialized to interface to the debug terminal. If these ports are reprogrammed, the terminal characteristics must be modified to suit, or the ports should be restored to the debugger-set characteristics prior to reinvoking the debugger.

Exception Vectors Used by 177Bug

The exception vectors used by the debugger are listed in Table B-3. These vectors must reside at the specified offsets in the target program's vector table for the associated debugger facilities (breakpoints, trace mode, etc.) to operate.

When the debugger handles one of the exceptions listed in Table B-3, the target stack pointer is left pointing past the bottom of the exception stack frame created; that is, it reflects the system stack pointer values just before the exception occurred. In this way, the operation of the debugger facility (through an exception) is transparent to users.

Table B-3. Exception Vectors Used by 177Bug

Vector Offset	Exception	177Bug Facility
$10	Illegal instruction	Breakpoints (used by **GO**, **GN**, **GT**)
$24	Trace	Trace operations (such as **T**, **TC**, **TT**)
$80-$B8	TRAP #0 - #14	Used internally
$BC	TRAP #15	System calls
$Note	Level 7 interrupt	ABORT push-button
$Note	Level 7 interrupt	AC Fail
$DC	FP Unimplemented Data Type	Software emulation and data type conversion of floating point data.
Note: Offsets marked "Note" depend on what the Vector Base Register (VBR) is set to in the VMEchip2.		

Example: Trace one instruction using debugger.

```
177Bug>RD
PC   =000E0000 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
SSP* =00010000 USP  =00010000 SFC  =1=UD  DFC  =1=UD
CACR =00000000=D:....._B:..._I:...           PCR  =04300000
D0   =00000000 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =000C0000 A6   =00000000 A7   =00010000    0000E
000  4E71            NOP                                  177Bug>T
PC   =00010006 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =0000DFFC MSP  =0000EFFC ISP* =0000FFFC SFC  =0=F0
DFC  =0=F0     CACR =0=........
D0   =00000001 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =0000FFFC
00010006 D280            ADD.L      D0,D1
177Bug>
```

Notice that the value of the target stack pointer register (A7) has not changed even though a trace exception has taken place. Your program may either use the exception vector table provided by 177Bug or it may create a separate exception vector table of its own. The two following sections detail these two methods.

Using 177Bug Target Vector Table

The 177Bug initializes and maintains a vector table area for target programs. A target program is any program started by the bug:

- Manually with **GO** command
- Manually with **TR** type command
- Automatically with the **BO** command

The start address of this target vector table area is the base address of the debugger memory. This address loads into the target-state VBR at power-up or cold-start reset and can be observed by using the **RD** command to display the target-state registers immediately after power-up.

The 177Bug initializes the target vector table with the debugger vectors listed in Table B-3 and fills the other vector locations with the address of a generalized exception handler (refer to the *177Bug Generalized Exception Handler* section in this chapter). The target program may take over as many vectors as desired by simply writing its own exception vectors into the table. If the vector locations listed in Table B-3 are overwritten, the accompanying debugger functions are lost.

The 177Bug maintains a separate vector table for its own use. In general, you do not have to be aware of the existence of the debugger vector table. It is completely transparent and you should never make any modifications to the vectors contained in it.

Creating a New Vector Table

Your program may create a separate vector table in memory to contain its exception vectors. If this is done, the program must change the value of the VBR to point to the new vector table. In order to use the debugger facilities you can copy the proper vectors from the 177Bug vector table into the corresponding vector locations in your program vector table.

The vector for the 177Bug generalized exception handler (described in detail in the *177Bug Generalized Exception Handler* section in this chapter) may be copied from offset $08 (bus error vector) in the target vector table to all locations in your program vector table where a separate exception handler is not used. This provides diagnostic support in the event that your program is stopped by an unexpected exception. The generalized exception handler gives a formatted display of the target registers and identifies the type of the exception.

The following is an example of a routine which builds a separate vector table and then moves the VBR to point at it:

```
*
***  BUILDX - Build exception vector table ****
*
BUILDX  MOVEC.L  VBR,A0            Get copy of VBR.
        LEA      $10000,A1         New vectors at $10000.
        MOVE.L   $80(A0),D0        Get generalized exception vector.
        MOVE.W   $3FC,D1           Load count (all vectors).
LOOP    MOVE.L   D0,(A1,D1)        Store generalized exception vector.
        SUBQ.W   #4,D1
        BNE.B    LOOP              Initialize entire vector table.
        MOVE.L   $10(A0),$10(A1)   Copy breakpoints vector.
        MOVE.L   $24(A0),$24(A1)   Copy trace vector.
        MOVE.L   $BC(A0),$BC(A1)   Copy system call vector.
        LEA.L    COPROCC(PC),A2    Get your exception vector.
        MOVE.L   A2,$2C(A1)        Install as F-Line handler.
        MOVEC.L  A1,VBR            Change VBR to new table.
        RTS
        END
```

It may happen that your program uses one or more of the exception vectors that are required for debugger operation. Debugger facilities may still be used, however, if your exception handler can determine when to handle the exception itself and when to pass the exception to the debugger.

When an exception occurs which you want to pass on to the debugger; i.e., **ABORT**, your exception handler must read the vector offset from the format word of the exception stack frame. This offset is added to the address of the 177Bug target program vector table (which your program saved), yielding the address of the 177Bug exception vector. The program then jumps to the address stored at this vector location, which is the address of the 177Bug exception handler.

Your program must make sure that there is an exception stack frame in the stack, and that it is exactly the same as the processor would have created for the particular exception before jumping to the address of the exception handler.

The following is an example of an exception handler which can pass an exception along to the debugger:

```
*
***  EXCEPT - Exception handler  ****
*
EXCEPT  SUBQ.L   #4,A7              Save space in stack for a PC value.
        LINK     A6,#0              Frame pointer for accessing PC space.
        MOVEM.L  A0-A5/D0-D7,-(SP)  Save registers.
        :
        : decide here if your code handles exception, if so, branch...
        :
        MOVE.L   BUFVBR,A0          Pass exception to debugger; Get saved VBR.
        MOVE.W   14(A6),D0          Get the vector offset from stack frame.
        AND.W    #$0FFF,D0          Mask off the format information.
        MOVE.L   (A0,D0.W),4(A6)    Store address of debugger exc handler.
        MOVEM.L  (SP)+,A0-A5/D0-D7  Restore registers.
        UNLK     A6
        RTS                         Put addr of exc handler into PC and go.
```

177Bug Generalized Exception Handler

The 177Bug has a generalized exception handler which it uses to handle all of the exceptions not listed in Table B-3. For all these exceptions, the target stack pointer is left pointing to the top of the exception stack frame created. In this way, if an unexpected exception occurs during execution of your code, you are presented with the exception stack frame to help determine the cause of the exception. The following example illustrates this:

Example:

Bus error at address $F00000. It is assumed for this example that an access of memory location $F00000 initiates bus error exception processing.

```
177Bug>RD
PC   =000E0000 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
SSP* =00010000 USP  =00010000 SFC  =1=UD  DFC  =1=UD
CACR =00000000=D:....._B:..._I:...           PCR  =04300000
D0   =00000000 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =000C0000 A6   =00000000 A7   =00010000       0000E
000  4E71             NOP                                    177Bug>T

Exception: Access Fault (Local Off Board)
PC =FF839154 SR =2704
Format/Vector =7008
SSW =0145 Fault Address =00F00000 Effective Address =0000D4E8
PC   =00010000 SR   =2708=TR:OFF_S._7_.N...  VBR  =00000000
USP  =0000DFFC MSP  =0000EFFC ISP* =0000FFFC SFC  =0=F0
DFC  =0=F0     CACR =0=........
D0   =00000001 D1   =00000001 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000002 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =0000FFC0
00010000 203900F0 0000      MOVE.L      ($F00000).L,D0
177Bug>
```

Notice that the target stack pointer is different. The target stack pointer now points to the last value of the exception stack frame that was stacked. The exception stack frame may now be examined using the **MD** command.

```
177Bug>MD (A7):&30
0000FFC0 2708 0001 0000 7008  0000 FFFC 0105 0005   '.....p.........
0000FFD0 0005 0005 00F0 0000  0000 0A64 0000 FFF4   ...........d....
0000FFE0 00F0 0000 FFFF FFFF  00F0 0000 FFFF FFFF   ................
0000FFF0 2708 0001 A708 0001  0000 0000             '...........
177Bug>
```

B

Floating Point Support

The floating point unit (FPU) of the MC68060 microprocessor chip is supported in 177Bug. For MVME177Bug, the commands:

- MD
- MM
- RM
- RS

have been extended to allow display and modification of floating point data in registers and in memory. Floating point instructions can be assembled/disassembled with the **DI** option of the **MD** and **MM** commands.

Valid data types that can be used when modifying a floating point data register or a floating point memory location:

Integer Data Types	
12	Byte
1234	Word
12345678	Longword

Floating Point Data Types	
1_FF_7FFFFF	Single Precision Real Format
1_7FF_FFFFFFFFFFFF	Double Precision Real Format
1_7FFF_FFFFFFFFFFFFFFFF	Extended Precision Real Format
1111_2103_123456789ABCDEF01	Packed Decimal Real Format
-3.12345678901234501_E+123	Scientific Notation Format (decimal)

When entering data in:

- Single precision
- Double precision
- Extended precision
- Packed decimal format

the following rules must be observed:

1. The sign field is the first field and is a binary field.
2. The exponent field is the second field and is a hexadecimal field.
3. The mantissa field is the last field and is a hexadecimal field.
4. The sign field, the exponent field, and at least the first digit of the mantissa field must be present (any unspecified digits in the mantissa field are set to zero).
5. Each field must be separated from adjacent fields by an underscore.
6. All the digit positions in the sign and exponent fields must be present.

Single Precision Real

This format would appear in memory as:

1-bit sign field	(1 binary digit)
8-bit biased exponent field	(2 hex digits. Bias = $7F)
23-bit fraction field	(6 hex digits)

A single precision number requires 4 bytes in memory.

Double Precision Real

This format would appear in memory as:

1-bit sign field	(1 binary digit)
11-bit biased exponent field	(3 hex digits. Bias = $3FF)
52-bit fraction field	(13 hex digits)

A double precision number requires 8 bytes in memory.

Note The single and double precision formats have an implied integer bit (always 1).

Extended Precision Real

This format would appear in memory as:

1-bit sign field	(1 binary digit)
15-bit biased exponent field	(4 hex digits. Bias = $3FFF)
64-bit mantissa field	(16 hex digits)

An extended precision number requires 10 bytes in memory.

Packed Decimal Real

This format would appear in memory as:

4-bit sign field	(4 binary digits)
16-bit exponent field	(4 hex digits)
68-bit mantissa field	(17 hex digits)

A packed decimal number requires 12 bytes in memory.

Scientific Notation

This format provides a convenient way to enter and display a floating point decimal number. Internally, the number is assembled into a packed decimal number then converted into a number of the specified data type.

Entering data in this format requires the following fields:

- An optional sign bit (+ or -)
- One decimal digit followed by a decimal point
- Up to 17 decimal digits (at least one must be entered)
- An optional Exponent field that consists of:
  - An optional underscore
  - The Exponent field identifier, letter "E"
  - An optional Exponent sign (+, -)
  - From 1 to 3 decimal digits

For more information about the MC68060 floating point unit, refer to the *MC68060 Microprocessor User's Manual*.

Additions to FLASH Commands

The 4MB of Flash memory on the MVME177 is unique in the way that the lower half "shadows" the PROM space in the memory map. A jumper on the board allows the user to switch visibility between the PROM sockets or 2MB (one half) of the Flash array.

Under software control, the lower half of Flash memory may be switched into and out of the address space normally occupied by the upper half of Flash memory, when the jumper (J8) is installed.

B

Flash Test Configuration Acceptable Entries

Command Input:

```
177-Diag>cf flash
FLASH Configuration Data:
Flash Device Test Mask =00000001 ? 0 or 1
Flash Test Starting Block =00000000 ? 0 through F
Flash Test Ending Block =0000000F ?0 through F
Save/Restore For PATS Test [Y?N] =Y ? Y or N
Fill Data =000000FF ? any byte 00 through FF
Test Data Increment/Decrement Step =00000001? 0, 1, 2, F (-1) etc.

177-Bug>sd
177-Diag>he flash
FLASH          Flash Memory Tests (DIR)
TESTS
ERASE          Erase
FILL           Fill
PATS           Patterns
```

Erase Test

The erase test erases Flash memory according to the current test configuration parameters selecting starting and ending blocks.

Command Input:

```
177-Diag>flash erase
```

Flash Fill Test

This test executes on the i28f008sa FLASHFILE™ memory devices, each having sixteen 64Kb blocks. On the MVME177 Flash memory is jumper selectable (mapping to begin at $FF800000 or $FFA00000).

The Flash Fill test fills Flash memory according to the current test configuration parameters selecting:

- Starting and ending block
- The data to fill with
- An increment/decrement value

Command Input:

```
177-Diag>flash fill
```

Flash Patterns Test

The Flash Patterns test writes and reads various data patterns in Flash memory according to the current test configuration parameters selecting starting and ending blocks, and saving / restoring of the Flash contents.

Note If you are running the Flash test and 177Bug itself resides in Flash, the test will fail as shown in the example below.

Command Input:

```
177-Diag>flash pats

177-Bug>sd
177-Diag>flash
FLASH TESTS:.................................Running->FAILED

FLASH/TESTS Test Failure Data:
```

Flash tests must be called individually and will only execute from PROM. ***On many boards 177Bug is running in Flash memory***.

B

Default Flash Test Configuration

Command Input:

```
177-Diag>cf flash
FLASH Configuration Data:
Flash Device Test Mask =00000001 ?
Flash Test Starting Block =00000000 ?
Flash Test Ending Block =0000000F ?
Save/Restore For PATS Test [Y?N] =Y ?
Fill Data =000000FF ?
Test Data Increment/Decrement Step =00000001?
```

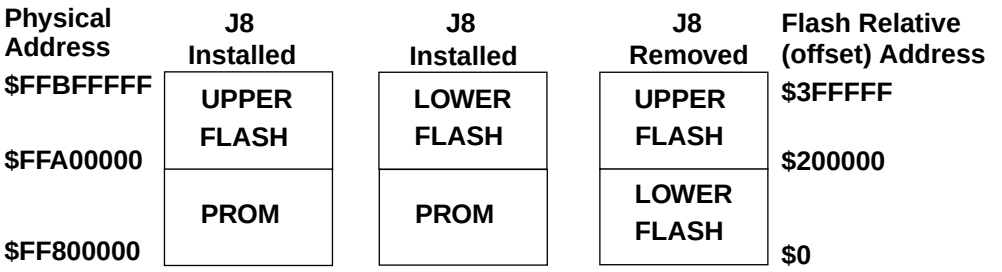


Figure B-1. Three Possible Mapping Options

When programming Flash memory on the MVME177, the destination starting address argument to the **PFLASH** command may be specified in two ways:

- A relative offset into the Flash memory array.
  - $0 = the bottom or lowest possible Flash memory array location.
  - $3FFFFF = the top location or end of the Flash memory array.
- An equivalent to the physical address that will apply when the entire Flash is mapped in (as when J8 is removed).
  - $FF800000 = the bottom or lowest possible Flash memory array location.
  - $FFBFFFFF = the top location or end of Flash memory.

These addressing methods apply no matter which of the three mapping options is in effect when the **PFLASH** command is entered.

Two sets of information are reported to the user when the **PFLASH** command is executed:

- The user is asked to verify the arguments entered.
- The block number and physical address for each operation (erasing, programming, and verifying) is displayed.

PFLASH reports the current physical address and the absolute block number for the operation in progress. In other words, the messages displayed while Flash is being:

- Programmed
- Erased
- Verified

may not appear to be based on the same destination starting address that was entered. This happens because the **PFLASH** command is needed to switch the portion of the Flash that is visible in order to program it. If switching is required, the map will be restored to the condition that existed before **PFLASH** was entered.

SFLASH Command

A new command is added to the 177BUG to assist the user in accessing the 4MB Flash memory array. However, the **SFLASH** command is valid only when jumper J8 is installed. **SFLASH** switches between the upper and lower half of the Flash array that appears in the visible address space. Two arguments are allowed:

- "L" to specifically select the lower half of Flash
- "U" for the upper half

If no argument is entered, the switch will simply change from the current half to the opposite and display a message to indicate the change.

```
177-Diag>sflash;l
FLASH Memory Visible Now = Lower Half

177-Diag>sflash;u
FLASH Memory Visible Now = Upper Half

177-Diag>sflash
FLASH Memory Visible Now = Lower Half
```

The destination address *DSAADR* of the **PFLASH** command is always interpreted using the J8 removed mapping, all of Flash mapped in. See Figure B-1, Three Possible Mapping Options.

This mapping is used whether or not J8 is installed and regardless of the most recent **SFLASH** command.

This mapping is also used if a relative offset is supplied as the destination address.

This treatment of the destination address allows the same **PFLASH** command to be used whether or not J8 is installed. It also allows programming the entire 4MB of flash or any portion with a single PFLASH command.

Assuming J8 is installed, then for this command sequence:

```
sflash;l
pflash ff800000 ff9fffff ffa00000
```

The **SFLASH;L** command has no effect on the destination address used by the **PFLASH** command. It is only a convenient way for the user to change which portion of the Flash memory array is in view.

In this case **PFLASH** programs the flash beginning at the location of physical address ffa00000 when J8 is out. This copies the bug from ROM at physical address ff800000 into the upper half of flash.

For the sequence:

```
pflash ff800000 ff9fffff 0
```

The **PFLASH** command programs the flash beginning at the Flash Relative Address of 0 (which corresponds to the flash physical address ff800000 when J8 is out). This copies the bug from ROM at physical address ff800000 into the lower half of flash.

Additionally, both of these **PFLASH** commands do the same thing when J8 is removed as they do when J8 is installed, regardless of any **SFLASH** command that may have been executed. In the case where the bug is programming over itself the user will not regain the bug prompt. Successful programming is indicated by the FAIL LED blinking twice per second. If the programming fails or if the programmed image is corrupted then it is likely that the bug image in lower flash is corrupted and the only recourse is to replace J8 and execute the bug from ROM.

For additional information on the **PFLASH** command consult the *Debugging Package for Motorola 68K CISC CPUs User's Manual*.

The 177Bug Debugger Command Set

The 177Bug debugger commands are summarized in Table B-4.

HE is the 177Bug help facility. **HE<CR>** displays only the command names of all available commands along with their appropriate titles. **HE** *COMMAND* displays:

- The command name
- Title for that particular command
- Complete command syntax

The command syntax is shown using the symbols explained earlier in this appendix.

Appendix C lists:

- Controllers
- Devices
- LUNs associated with the devices

The **CNFG** and **ENV** commands are explained in Appendix D. All other details of these two commands are explained in the *177Bug Diagnostics User's Manual*.

Details of all the other debugger commands are explained in the *Debugging Package for Motorola 68K CISC CPUs User's Manual*.

Table B-4. Debugger Commands

Command Mnemonic	Title	Command Line Syntax
AB	Automatic Bootstrap Operating System	**AB** [**;V**]
NOAB	No Autoboot	**NOAB**
AS	One Line Assembler	**AS** *ADDR*
BC	Block of Memory Compare	**BC** *RANGE DEL ADDR* [**; B** | **W** | **L**]
BF	Block of Memory Fill	**BF** *RANGE DEL data* [*DEL increment*] [**; B** | **W** | **L**]
BH	Bootstrap Operating System and Halt	**BH** [*DEL Controller LUN*][*DEL Device LUN*][*DEL String*]
BI	Block of Memory Initialize	**BI** *RANGE* [**;B** | **W** | **L**]
BM	Block of Memory Move	**BM** *RANGE DEL ADDR* [**; B** | **W** | **L**]
BO	Bootstrap Operating System	**BO** [*DEL Controller LUN*][*DEL Device LUN*][*DEL String*]
BR	Breakpoint Insert	**BR** [*ADDR*[:*COUNT*]]
NOBR	Breakpoint Delete	**NOBR** [*ADDR*]
BS	Block of Memory Search	**BS** *RANGE DEL TEXT* [**;B** | **W** | **L**] or **BS** *RANGE DEL data* [*DEL mask*] [**;B** | **W** | **L** [**,N**][**,V**]]
BV	Block of Memory Verify	**BV** *RANGE DEL data* [*increment*] [**;B** | **W** | **L**]
CM	Concurrent Mode	**CM** [[*PORT*][*DEL ID-STRING*][*DEL BAUD*] [*DEL PHONE-NUMBER*]] | [**;A**] | [**;H**]
NOCM	No Concurrent Mode	**NOCM**
CNFG	Configure Board Information Block	**CNFG** [**;**[**I**][**M**]]
CS	Checksum	**CS** *RANGE* [**;B** | **W** | **L**]
DC	Data Conversion	**DC** *EXP* | *ADDR* [**;**[**B**][**O**][**A**]]
DMA	DMA Block of Memory Move	**DMA** *RANGE DEL ADDR DEL VDIR DEL AM DEL BLK* [**;B** | **W** | **L**]
DS	One Line Disassembler	**DS** *ADDR* [**:***COUNT* | *DEL ADDR*]
DU	Dump S-records	**DU** [*PORT*]*DEL RANGE* [*DEL TEXT*][*DEL ADDR*] [*DEL OFFSET*][**;B** | **W** | **L**]
ECHO	Echo String	**ECHO** [*PORT*]*DEL*{*hexadecimal number*} {'*string*'}

Table B-4. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
ENV	Set Environment to Bug/Operating System	**ENV** [**;**[**D**]]
GD	Go Direct (Ignore Breakpoints)	**GD** [*ADDR*]
GN	Go to Next Instruction	**GN**
GO	Go Execute User Program	**GO** [*ADDR*]
GT	Go to Temporary Breakpoint	**GT** *ADDR*
HE	Help	**HE** [*COMMAND*]
IOC	I/O Control for Disk	**IOC**
IOI	I/O Inquiry	**IOI** [**;**[**C** | **L**]]
IOP	I/O Physical (Direct Disk Access)	**IOP**
IOT	I/O "TEACH" for Configuring Disk Controller	**IOT** [**;**[**A**][**F**][**H**][**T**]]
IRQM	Interrupt Request Mask	**IRQM** [*MASK*]
LO	Load S-records from Host	**LO** [*n*] [*ADDR*] [**;X** | **C** | **T**] [*=text*]
MA	Macro Define/Display	**MA** [*NAME* | **;L**]
NOMA	Macro Delete	**NOMA** [*NAME*]
MAE	Macro Edit	**MAE** *name line#* [*string*]
MAL	Enable Macro Expansion Listing	**MAL**
NOMAL	Disable Macro Expansion Listing	**NOMAL**
MAW	Save Macros	**MAW** [*controller LUN*][*DEL*[*device LUN*][*DEL block #*]]
MAR	Load Macros	**MAR** [*controller LUN*][*DEL*[*device LUN*][*DEL block #*]]
MD	Memory Display	**MD**[**S**] *ADDR*[**:***COUNT* | *ADDR*] [**;** [**B** | **W** | **L** | **S** | **D** | **X** | **P** | **DI**]]
MENU	Menu	**MENU**
MM	Memory Modify	**MM** *ADDR*[**;**[[**B** | **W** | **L** | **S** | **D** | **X** | **P**][**A**][**N**]] | [**DI**]]

Table B-4. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
MMD	Memory Map Diagnostic	**MMD** *RANGE DEL increment*[**;B** | **W** | **L**]
MS	Memory Set	**MS** *ADDR* {*Hexadecimal number*} {'*string*'}
MW	Memory Write	**MW** *ADDR DATA* [**;B** | **W** | **L**]
NAB	Automatic Network Boot Operating System	**NAB**
NBH	Network Boot Operating System and Halt	**NBH** [*Controller LUN*][*Device LUN*][*Client IP Address*] [*Server IP Address*][*String*]
NBO	Network Boot Operating System	**NBO** [*Controller LUN*][*Device LUN*][*Client IP Address*] [*Server IP Address*][*String*]
NIOC	Network I/O Control	**NIOC**
NIOP	Network I/O Physical	**NIOP**
NIOT	Network I/O Teach	**NIOT** [**;**[**H**] | [**A**]]
NPING	Network Ping	**NPING** *Controller-LUN Device-LUN Source-IP Destination-IP* [*N-Packets*]
OF	Offset Registers Display/Modify	**OF** [**R***n*[**;A**]]
PA	Printer Attach	**PA** [*n*]
NOPA	Printer Detach	**NOPA** [*n*]
PF	Port Format	**PF** [*PORT*]
NOPF	Port Detach	**NOPF** [*PORT*]
PFLASH	Load FLASH Memory	**PFLASH** *SSADDR SEADDR DSADDR* [*IEADDR*] [; [**A** | **R**] [**X**]] **PFLASH** *SSADDR SEADDR DSADDR* [*IEADDR*] [;[**B** | **W** | **L**] [**A** | **R**] [**X**]]
PS	Put RTC Into Power Save Mode for Storage	**PS**
RB	ROMboot Enable	**RB**[**;V**]
NORB	ROMboot Disable	**NORB**
RD	Register Display	**RD** {[+ | **-** | =][*DNAME*][/]} {[+ | **-** | =][*REG1*[**-** *REG2*]][/]} [**;E**]

Table B-4. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
REMOTE	Connect the Remote Modem to CSO	**REMOTE**
RESET	Cold/Warm Reset	**RESET**
RL	Read Loop	**RL** *ADDR*;[**B** | **W** | **L**]
RM	Register Modify	**RM** [*REG*] [;[**S** | **D**]]
RS	Register Set	**RS** *REG* [*DEL EXP* | *DEL ADDR*][;[**S** | **D**]]
SD	Switch Directories	**SD**
SET	Set Time and Date	**SET** *mmddyyhhmm* or **SET** *n*;**C**
SFLASH	Switch Visible FLASH	**SFLASH** [;**L** | **U**]
SYM	Symbol Table Attach	**SYM** [*ADDR*]
NOSYM	Symbol Table Detach	**NOSYM**
SYMS	Symbol Table Display/Search	**SYMS** [*symbol-name*] | [;**S**]
T	Trace	**T** [*COUNT*]
TA	Terminal Attach	**TA** [*port*]
TC	Trace on Change of Control Flow	**TC** [*count*]
TIME	Display Time and Date	**TIME** [;[**C** | **L** | **O**]]
TM	Transparent Mode	**TM** [*n*] [*ESCAPE*]
TT	Trace to Temporary Breakpoint	**TT** *ADDR*
VE	Verify S-Records Against Memory	**VE** [*n*] [*ADDR*] [;[**X**][**C**]] [=*text*]
VER	Display Revision/Version	**VER** [; **E**]
WL	Write Loop	**WL** *ADDR*:***DATA***;[**B** | **W** | **L**]

Disk/Tape Controller Data

C

Disk/Tape Controller Modules Supported

The following VMEbus disk/tape controller modules are supported by 177Bug. The default address for each controller type is First Address. The controller can be addressed by First CLUN during:

- Command **BH**
- Command **BO**
- Command **IOP**
- TRAP #15 calls .DSKRD or .DSKWR

Note that if another controller of the same type is used, the second one must have its address changed by its onboard jumpers and/or switches, so that it matches Second Address and can be called up by Second CLUN.

Controller Type	First CLUN	First Address	Second CLUN	Second Address
CISC Single Board Computer (SBC)	$00 (Note 1)	--	--	--
MVME320 - Winchester/Floppy Controller	$11 (Note 2)	$FFFFB000	$12 (Note 2)	$FFFFA000
MVME323 - ESDI Winchester Controller	$08	$FFFFA000	$09	$FFFFA200
MVME327A - SCSI Controller	$02	$FFFFA600	$03	$FFFFA700
MVME328 - SCSI Controller	$06	$FFFF9000	$07	$FFFF9800
MVME328 - SCSI Controller	$16	$FFFF4800	$17	$FFFF5800
MVME328 - SCSI Controller	$18	$FFFF7000	$19	$FFFF7800
MVME350 - Streaming Tape Controller	$04	$FFFF5000	$05	$FFFF5100

Notes: (1) If the SBC (e.g., an MVME177) SCSI port is used, then the SBC module has CLUN 0.

(2) For SBCs, the first MVME320 has CLUN $11, and the second MVME320 has CLUN $12.

Disk/Tape Controller Default Configurations

Note SCSI Common Command Set (CCS) devices are only the ones tested by Motorola Computer Group.

CISC Embedded Controllers --Seven Devices

Controller LUN	Address	Device LUN	Device Type
0	$*xxxxxxxx*	00 10 20 30 40 50 60	SCSI Common Command Set (CCS), which may be any of these: - Fixed direct access - Removable flexible direct access (TEAC style) - CD-ROM - Sequential access

MVME320 --Four Devices

Controller LUN	Address	Device LUN	Device Type
11	$FFFFB000	0	Winchester hard drive
		1	Winchester hard drive
12	$FFFFAC00	2	5-1/4" DS/DD 96 TPI floppy drive
		3	5-1/4" DS/DD 96 TPI floppy drive

MVME323 -- Four Devices

Controller LUN	Address	Device LUN	Device Type
8	$FFFFA000	0	ESDI Winchester hard drive
		1	ESDI Winchester hard drive
9	$FFFFA200	2	ESDI Winchester hard drive
		3	ESDI Winchester hard drive

MVME327A --Nine Devices

Controller LUN	Address	Device LUN	Device Type
2	$FFFFA600	00 10 20 30 40 50 60	SCSI Common Command Set (CCS), which may be any of these: - Fixed direct access - Removable flexible direct access (TEAC style) - CD-ROM - Sequential access
3	$FFFFA700		
		80	Local floppy drive
		81	Local floppy drive

MVME328 -- Fourteen Devices

Controller LUN	Address	Device LUN	Device Type
6	$FFFF9000	00 08 10 18 20 28 30	SCSI Common Command Set (CCS), which may be any of these: - Removable flexible direct access (TEAC style) - CD-ROM - Sequential access
7	$FFFF9800		
16	$FFFF4800		
17	$FFFF5800	40 48 50 58 60 68 70	Same as above, but these will only be available if the daughter card for the second SCSI channel is present.
18	$FFFF7000		
19	$FFFF7800		

MVME350 --One Device

Controller LUN	Address	Device LUN	Device Type
4	$FFFF5000	0	QIC-02 streaming tape drive
5	$FFFF5100		

IOT Command Parameters for Supported Floppy Types

The following table lists the proper **IOT** command parameters for floppies used with boards such as the:

- MVME328
- MVME167
- MVME177
- MVME187

IOT Parameter	Floppy Types and Formats						
	DSDD5	PCXT8	PCXT9	PCXT9_3	PCAT	PS2	SHD
Sector Size 0- 128 1- 256 2- 512 3-1024 4-2048 5-4096	1	2	2	2	2	2	2
Block Size: 0- 128 1- 256 2- 512 3-1024 4-2048 5-4096	1	1	1	1	1	1	1
Sectors/Track	10	8	9	9	F	12	24
Number of Heads	2	2	2	2	2	2	2
Number of Cylinders	50	28	28	50	50	50	50
Precomp. Cylinder	50	28	28	50	50	50	50
Reduced Write Current Cylinder	50	28	28	50	50	50	50
Step Rate Code	0	0	0	0	0	0	0
Single/Double DATA Density	D	D	D	D	D	D	D
Single/Double TRACK Density	D	D	D	D	D	D	D
Single/Equal_in_all Track Zero Density	S	E	E	E	E	E	E
Slow/Fast Data Rate	S	S	S	S	F	F	F

IOT Parameter	Floppy Types and Formats (Continued)						
	DSDD5	PCXT8	PCXT9	PCXT9_3	PCAT	PS2	SHD
Other Characteristics							
Number of Physical Sectors	0A00	0280	02D0	05A0	0960	0B40	1680
Number of Logical Blocks (100 in size)	09F8	0500	05A0	0B40	12C0	1680	2D00
Number of Bytes in Decimal	653312	327680	368460	737280	1228800	1474560	2949120
Media Size/Density	5.25/DD	5.25/DD	5.25/DD	3.5/DD	5.25/D	3.5/HD	3.5/ED

Notes:
1. All numerical parameters are in hexadecimal unless otherwise noted.
2. The DSDD5 type floppy is the default setting for the debugger.

Configure and Environment Commands

D

Configure Board Information Block

CNFG [;[I][M]]

This command is used to display and configure the board information block. This block resides within the Non-Volatile RAM (NVRAM). Refer to the *Single Board Computer Programmer's Reference Guide* for the actual location. The information block contains various elements detailing specific operation parameters of the hardware. The *Single Board Computer Programmer's Reference Guide:*

- Describes the elements within the board information block
- Lists the size of each element
- Lists the logical offset of each element

The **CNFG** command does *not* describe the elements and their use. The board information block contents are checksummed for validation purposes. This checksum is the last element of the block.

Example: to display the current contents of the board information block.

```
177-Bug>cnfg
Board (PWA) Serial Number = "000000061050"
Board Identifier          = "MVME177-001 "
Artwork (PWA) Identifier  = "01-W3944B01D "
MPU Clock Speed           = "5000"
Ethernet Address          = 08003E20A867
Local SCSI Identifier     = "07"
Optional Board 1 Artwork (PWA) Identifier  = "        "
Optional Board 1 (PWA) Serial Number       = "        "
Optional Board 2 Artwork (PWA) Identifier  = "        "
Optional Board 2 (PWA) Serial Number       = "        "
177-Bug>
```

Note that the parameters that are quoted are left-justified character (ASCII) strings padded with space characters, and the quotes (") are displayed to indicate the size of the string. Parameters that are not quoted are considered data strings, and data strings are right-justified. The data strings are padded with zeroes if the length is not met.

In the event of corruption of the board information block, the command displays a question mark "?" for nondisplayable characters.The following warning message:

```
(WARNING: Board Information Block Checksum Error)
```

is also displayed in the event of a checksum failure.

Using the **I** option initializes the unused area of the board information block to zero.

Modification is permitted by using the **M** option of the command. At the end of the modification session, you are prompted for the update to Non-Volatile RAM (NVRAM). A "Y" response must be made for the update to occur; any other response terminates the update (disregards all changes). The update also recalculates the checksum.

Be cautious when modifying parameters. Some of these parameters are set up by the factory, and correct board operation relies upon these parameters.

Once modification/update is complete, you can now display the current contents as described earlier.

Set Environment to Bug/Operating System

ENV [;[D]]

The **ENV** command allows you to interactively view/configure all Bug operational parameters that are kept in Battery Backed Up RAM (BBRAM), also known as Non-Volatile RAM (NVRAM). The operational parameters are saved in NVRAM and used whenever power is lost.

Any time the Bug uses a parameter from NVRAM, the NVRAM contents are first tested by checksum to insure the integrity of the NVRAM contents. In the instance of BBRAM checksum failure, certain default values are assumed as stated below.

The bug operational parameters (which are kept in NVRAM) are not initialized automatically on power up/warm reset. It is up to the Bug user to invoke the **ENV** command. Once the **ENV** command is invoked and executed without error, Bug default and/or user parameters are loaded into NVRAM along with checksum data. If any of the operational parameters have been modified, these new parameters will not be in effect until a reset/powerup condition.

If the **ENV** command is invoked with no options on the command line, you are prompted to configure all operational parameters. If the **ENV** command is invoked with the option **D**, ROM defaults will be loaded into NVRAM.

The 177Bug **ENV** parameter display is shown in the following example:

```
177-Bug>env
Bug or System environment [B/S] = S?
Field Service Menu Enable [Y/N] = Y?
Remote Start Method Switch [G/M/B/N] = B?
Probe System for Supported I/O Controllers [Y/N] = Y?
Negate VMEbus SYSFAIL* Always [Y/N] = N?
Local SCSI Bus Reset on Debugger Startup [Y/N] = N?
Local SCSI Bus Negotiations Type [A/S/N]       = A?
Ignore CFGA Block on a Hard Disk Boot [Y/N]    = Y?
```

D

```
Auto Boot Enable [Y/N]   = N?
Auto Boot at power-up only [Y/N] = Y?
Auto Boot Controller LUN = 00?
Auto Boot Device LUN     = 00?
Auto Boot Abort Delay    = 15?
Auto Boot Default String [NULL for a empty string] = ?
ROM Boot Enable [Y/N]            = N?
ROM Boot at power-up only [Y/N]  = Y?
ROM Boot Enable search of VMEbus [Y/N] = N?
ROM Boot Abort Delay             = 0?
ROM Boot Direct Starting Address = FF800000?
ROM Boot Direct Ending Address   = FFBFFFFC?
Network Auto Boot Enable [Y/N]   = N?
Network Auto Boot at power-up only [Y/N] = Y?
Network Auto Boot Controller LUN = 00?
Network Auto Boot Device LUN     = 00?
Network Auto Boot Abort Delay    = 5?
Network Auto Boot Configuration Parameters Pointer (NVRAM) = 00000000?
Memory Search Starting Address   = 00000000?
Memory Search Ending Address     = 02000000?
Memory Search Increment Size     = 00010000?
Memory Search Delay Enable [Y/N] = N?
Memory Search Delay Address      = FFFFCE0F?
Memory Size Enable [Y/N]         = Y?
Memory Size Starting Address     = 00000000?
Memory Size Ending Address       = 02000000?
Base Address of Local Memory     = 00000000?
Size of Local Memory Board #0    = 02000000?
Size of Local Memory Board #1    = 00000000?
Slave Enable #1 [Y/N] = Y?
Slave Starting Address #1 = 00000000?
Slave Ending Address #1   = 01FFFFFF?
Slave Address Translation Address #1 = 00000000?
Slave Address Translation Select #1  = 00000000?
Slave Control #1 = 03FF?
Slave Enable #2 [Y/N] = Y?
Slave Starting Address #2 = FFE00000?
Slave Ending Address #2   = FFE1FFFF?
Slave Address Translation Address #2 = 00000000?
Slave Address Translation Select #2  = 00000000?
Slave Control #2 = 01EF?
Master Enable #1 [Y/N] = Y?
Master Starting Address #1 = 02000000?
Master Ending Address #1   = EFFFFFFF?
Master Control #1 = 0D?
Master Enable #2 [Y/N] = N?
```

```
Master Starting Address #2 = 00000000?
Master Ending Address #2   = 00000000?
Master Control #2 = 00?
Master Enable #3 [Y/N] = N?
Master Starting Address #3 = 00000000?
Master Ending Address #3   = 00000000?
Master Control #3 = 00?
Master Enable #4 [Y/N] = N?
Master Starting Address #4 = 00000000?
Master Ending Address #4   = 00000000?
Master Address Translation Address #4 = 00000000?
Master Address Translation Select #4  = 00000000?
Master Control #4 = 00?
Short I/O (VMEbus A16) Enable [Y/N] = Y?
Short I/O (VMEbus A16) Control      = 01?
F-Page (VMEbus A24) Enable [Y/N]    = Y?
F-Page (VMEbus A24) Control   = 02?
ROM Speed Bank A Code         = 05?
ROM Speed Bank B Code         = 05?
Static RAM Speed Code         = 01?
PCC2 Vector Base              = 05?
VMEC2 Vector Base #1          = 06?
VMEC2 Vector Base #2          = 07?
VMEC2 GCSR Group Base Address = D4?
VMEC2 GCSR Board Base Address = 00?
VMEbus Global Time Out Code   = 01?
Local Bus Time Out Code       = 00?
VMEbus Access Time Out Code   = 02?
177-Bug>
```

The **ENV** command parameters to be configured are explained in the following table:

D

Table D-1. ENV Command Parameters

ENV Parameter and Options	Default	Meaning of Default
Bug or System environment [B/S]	S	System mode
Field Service Menu Enable [Y/N]	Y	Display field service menu.
Remote Start Method Switch [G/M/B/N]	B	Use both the Global Control and Status Register (GCSR) in the VMEchip2, and the Multiprocessor Control Register (MPCR) in shared RAM, methods to pass and start execution of cross-loaded program.
Probe System for Supported I/O Controllers [Y/N]	Y	Accesses will be made to VMEbus to determine presence of supported controllers.
Negate VMEbus SYSFAIL* Always [Y/N]	N	Negate VMEbus SYSFAIL after successful completion or entrance into the bug command monitor.
Local SCSI Bus Reset on Debugger Startup [Y/N]	N	Local SCSI bus is not reset on debugger startup.
Local SCSI Bus Negotiations Type [A/S/N]	A	Asynchronous
Ignore CFGA Block on a Hard Disk Boot [Y/N]	Y	Enable the ignorance of the Configuration Area (CFGA) Block (hard disk only).
Auto Boot Enable [Y/N]	N	Auto Boot function is disabled.
Auto Boot at power-up only [Y/N]	Y	Auto Boot is attempted at power up reset only.
Auto Boot Controller LUN	00	LUN of a disk/tape controller module currently supported by the Bug. Default is $0.
Auto Boot Device LUN	00	LUN of a disk/tape device currently supported by the Bug. Default is $0.
Auto Boot Abort Delay	15	This is the time in seconds that the Auto Boot sequence will delay before starting the boot. The purpose for the delay is to allow you the option of stopping the boot by use of the Break key. The time value is from 0 through 255 seconds.

Table D-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Auto Boot Default String [Y(NULL String)/(String)]		You may specify a string (filename) which is passed on to the code being booted. Maximum length is 16 characters. Default is the null string.
ROM Boot Enable [Y/N]	N	ROMboot function is disabled.
ROM Boot at power-up only [Y/N]	Y	ROMboot is attempted at power up only.
ROM Boot Enable search of VMEbus [Y/N]	N	VMEbus address space will not be accessed by ROMboot.
ROM Boot Abort Delay	00	This is the time in seconds that the ROMboot sequence will delay before starting the boot. The purpose for the delay is to allow you the option of stopping the boot by use of the Break key. The time value is from 0 through 255 seconds.
ROM Boot Direct Starting Address	FF800000	First location tested when the Bug searches for a ROMboot Module.
ROM Boot Direct Ending Address	FFBFFFFC	Last location tested when the Bug searches for a ROMboot Module.
Network Auto Boot Enable [Y/N]	N	Network Auto Boot function is disabled.
Network Auto Boot at power-up only [Y/N]	Y	Network Auto Boot is attempted at power up reset only.
Network Auto Boot Controller LUN	00	LUN of a disk/tape controller module currently supported by the Bug. Default is $0.
Network Auto Boot Device LUN	00	LUN of a disk/tape device currently supported by the Bug. Default is $0.
Network Auto Boot Abort Delay	5	This is the time in seconds that the Network Boot sequence will delay before starting the boot. The purpose for the delay is to allow you the option of stopping the boot by use of the Break key. The time value is from 0 through 255 seconds.

Table D-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Network Autoboot Configuration Parameters Pointer (NVRAM)	00000000	This is the address where the network interface configuration parameters are to be saved/retained in NVRAM; these parameters are the necessary parameters to perform an unattended network boot. **Caution** If you use the **NIOT** debugger command, these parameters need to be saved/retained in the NVRAM, somewhere in the address range $FFFC0000 through $FFFC0FFF. The **NIOT** parameters do not exceed 128 bytes in size. The location for these parameters is determined by setting this **ENV** pointer. If you have used the exact same space for your own program information or commands, they will be overwritten and lost. You can relocate the network interface configuration parameters in this space by using the **ENV** command to change the `Network Auto Boot Configuration Parameters Pointer (NVRAM)` from its default of `00000000` to the value you need so as to be clear of your data within NVRAM.

Table D-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Memory Search Starting Address	00000000	Where the Bug begins to search for a work page (a 64KB block of memory) to use for vector table, stack, and variables. This must be a multiple of the debugger work page, modulo $10000 (64KB). In a multi-MVME177 environment, each MVME177 board could be set to start its work page at a unique address to allow multiple debuggers to operate simultaneously.
Memory Search Ending Address	02000000	Top limit of the Bug's search for a work page. If a contiguous block of memory, 64KB in size, is not found in the range specified by Memory Search Starting Address and Memory Search Ending Address parameters, then the bug will place its work page in the onboard static RAM on the MVME177. Default Memory Search Ending Address is the calculated size of local memory.
Memory Search Increment Size	00010000	This multi-CPU feature is used to offset the location of the Bug work page. This must be a multiple of the debugger work page, modulo $10000 (64KB). Typically, Memory Search Increment Size is the product of CPU number and size of the Bug work page. Example: first CPU $0 (0 x $10000), second CPU $10000 (1 x $10000), etc.
Memory Search Delay Enable [Y/N]	N	There will be no delay before the Bug begins its search for a work page.

Table D-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Memory Search Delay Address	FFFFCE0F	Default address is $FFFFCE0F. This is the MVME177 GCSR GPCSR0 as accessed through VMEbus A16 space and assumes the MVME177 GRPAD (group address) and BDAD (board address within group) switches are set to "on". This byte-wide value is initialized to $FF by MVME177 hardware after a System or Power-on Reset. In a multi-MVME177 environment, where the work pages of several Bugs are to reside in the memory of the primary (first) MVME177, the non-primary CPUs will wait for the data at the Memory Search Delay Address to be set to $00, $01, or $02 (refer to the *Memory Requirements* section in Chapter 3 for the definition of these values) before attempting to locate their work page in the memory of the primary CPU.
Memory Size Enable [Y/N]	Y	Memory will be sized for Self Test diagnostics.
Memory Size Starting Address	00000000	Default Starting Address is $0.
Memory Size Ending Address	02000000	Default Ending Address is the calculated size of local memory.
Base Address of Local Memory	00000000	Beginning address of Local Memory. It must be a multiple of the Local Memory board size, starting with 0. The Bug will set up hardware address decoders so that Local Memory resides as one contiguous block at this address. Default is $0.
Size of Local Memory Board #0 Size of Local Memory Board #1	02000000 00000000	You are prompted twice, once for each possible MVME177 memory board. Default is the calculated size of the memory board.

Table D-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Slave address decoders setup. The slave address decoders are use to allow another VMEbus master to access a local resource of the MVME177. There are two slave address decoders set. They are set up as follows:		
Slave Enable #1 [Y/N]	Y	Yes, set up and enable the Slave Address Decoder #1.
Slave Starting Address #1	00000000	Base address of the local resource that is accessible by the VMEbus. Default is the base of local memory, $0.
Slave Ending Address #1	01FFFFFF	Ending address of the local resource that is accessible by the VMEbus. Default is the end of calculated memory.
Slave Address Translation Address #1	00000000	This register will allow the VMEbus address and the local address to be different. The value in this register is the base address of local resource that is associated with the starting and ending address selection from the previous questions. Default is 0.
Slave Address Translation Select #1	00000000	This register defines which bits of the address are significant. A logical one "1" indicates significant address bits, logical zero "0" is non-significant. Default is 0.
Slave Control #1	03FF	Defines the access restriction for the address space defined with this slave address decoder. Default is $01FF.
Slave Enable #2 [Y/N]	Y	Yes, set up and enable the Slave Address Decoder #2.
Slave Starting Address #2	FFE00000	Base address of the local resource that is accessible by the VMEbus. Default is the base address of static RAM, $FFE00000.

Table D-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Slave Ending Address #2	FFE1FFFF	Ending address of the local resource that is accessible by the VMEbus. Default is the end of static RAM, $FFE1FFFF.
Slave Address Translation Address #2	00000000	Works the same as Slave Address Translation Address #1. Default is 0.
Slave Address Translation Select #2	00000000	Works the same as Slave Address Translation Select #1. Default is 0.
Slave Control #2	01EF	Defines the access restriction for the address space defined with this slave address decoder. Default is $01EF.
Master Enable #1 [Y/N]	Y	Yes, set up and enable the Master Address Decoder #1.
Master Starting Address #1	02000000	Base address of the VMEbus resource that is accessible from the local bus. Default is the end of calculated local memory.
Master Ending Address #1	EFFFFFFF	Ending address of the VMEbus resource that is accessible from the local bus. Default is the end of calculated memory.
Master Control #1	0D	Defines the access characteristics for the address space defined with this master address decoder. Default is $0D.
Master Enable #2 [Y/N]	N	Do not set up and enable the Master Address Decoder #2.
Master Starting Address #2	00000000	Base address of the VMEbus resource that is accessible from the local bus. Default is $00000000.
Master Ending Address #2	00000000	Ending address of the VMEbus resource that is accessible from the local bus. Default is $00000000.
Master Control #2	00	Defines the access characteristics for the address space defined with this master address decoder. Default is $00.

Table D-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Master Enable #3 [Y/N]	N	Do not set up and enable the Master Address Decoder #3.
Master Starting Address #3	00000000	Base address of the VMEbus resource that is accessible from the local bus. Default is $0.
Master Ending Address #3	00000000	Ending address of the VMEbus resource that is accessible from the local bus. Default is $0.
Master Control #3	00	Defines the access characteristics for the address space defined with this master address decoder. Default is $0.
Master Enable #4 [Y/N]	N	Do not set up and enable the Master Address Decoder #4.
Master Starting Address #4	00000000	Base address of the VMEbus resource that is accessible from the local bus. Default is $0.
Master Ending Address #4	00000000	Ending address of the VMEbus resource that is accessible from the local bus. Default is $0.
Master Address Translation Address #4	00000000	This register will allow the VMEbus address and the local address to be different. The value in this register is the base address of VMEbus resource that is associated with the starting and ending address selection from the previous questions. Default is 0.
Master Address Translation Select #4	00000000	This register defines which bits of the address are significant. A logical one "1" indicates significant address bits, logical zero "0" is non-significant. Default is 0.
Master Control #4	00	Defines the access characteristics for the address space defined with this master address decoder. Default is $00.
Short I/O (VMEbus A16) Enable [Y/N]	Y	Yes, enable the Short I/O Address Decoder.

Table D-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Short I/O (VMEbus A16) Control	01	Defines the access characteristics for the address space defined with the Short I/O address decoder. Default is $01.
F-Page (VMEbus A24) Enable [Y/N]	Y	Yes, Enable the F-Page Address Decoder.
F-Page (VMEbus A24) Control	02	Defines the access characteristics for the address space defined with the F-Page address decoder. Default is $02.
ROM Speed Bank A Code ROM Speed Bank B Code	05 05	Used to set up the ROM speed. Default $05 = 165 ns (for 50 MHz MVME177s), or $05 = 132 ns (for 60 MHz MVME177s).
Static RAM Speed Code	01	Used to set up the SRAM speed. Default $01 = 125 ns (for 50 MHz MVME177s).
	00	Used to set up the SRAM speed. Default $00 = 132 ns (for 60 MHz MVME177s).
PCC2 chip Vector Base VMEC2 Vector Base #1 VMEC2 Vector Base #2	05 06 07	Base interrupt vector for the component specified. Default: PCC2 chip = $05, VMEchip2 Vector 1 = $06, VMEchip2 Vector 2 = $07.
VMEC2 GCSR Group Base Address	D4	Specifies the group address ($FFFFxx00) in Short I/O for this board. Default = $D4.
VMEC2 GCSR Board Base Address	00	Specifies the base address ($FFFFD4x0) in Short I/O for this board. Default = $00.
VMEbus Global Time Out Code	01	This controls the VMEbus timeout when systems controller. Default $01 = 64 µs.
Local Bus Time Out Code	00	This controls the local bus timeout. Default $00 =8 µs.
VMEbus Access Time Out Code	02	This controls the local bus to VMEbus access timeout. Default $02 = 32 ms.

Network Controller Data

E

Network Controller Modules Supported

The VMEbus network controller modules in the following table are supported by 177Bug. The default address for each type and position is shown to indicate where the controller must reside to be supported by 177Bug. The controllers are accessed via the specified CLUN and DLUNs listed here. The CLUN and DLUNs are used in conjunction with the debugger commands:

- NBH
- NBO
- NIOP
- NIOC
- NIOT
- NPING
- NAB

and also with the debugger system calls:

- .NETRD
- .NETWR
- .NETFOPN
- .NETFRD
- .NETCFIG
- .NETCTRL

Controller Type	CLUN	DLUN	Address	Interface Type
MVME177	$00	$00	$FFF46000	Ethernet
MVME376	$02	$00	$FFFF1200	Ethernet
MVME376	$03	$00	$FFFF1400	Ethernet
MVME376	$04	$00	$FFFF1600	Ethernet
MVME376	$05	$00	$FFFF5400	Ethernet
MVME376	$06	$00	$FFFF5600	Ethernet
MVME376	$07	$00	$FFFFA400	Ethernet
MVME374	$10	$00	$FF000000	Ethernet
MVME374	$11	$00	$FF100000	Ethernet
MVME374	$12	$00	$FF200000	Ethernet
MVME374	$13	$00	$FF300000	Ethernet
MVME374	$14	$00	$FF400000	Ethernet
MVME374	$15	$00	$FF500000	Ethernet

Troubleshooting the MVME177: Solving Startup Problems

F

- Try these simple troubleshooting steps before calling for help or sending your CPU board back for repair.
- Some of the procedures will return the board to the factory debugger environment. (The board was tested under these conditions before it left the factory.)
- Selftest may not run in all user-customized environments.

Table F-1. Basic Troubleshooting Steps

Condition ...	Possible Problem ...	Try This ...
I. Nothing works, no display on the terminal.	A. If the RUN or +12V LED is not lit, the board may not be getting correct power.	1. Make sure the system is plugged in. 2. Check that the board is securely installed in its backplane or chassis. 3. Check that all necessary cables are connected to the board, per this manual. 4. Check for compliance with System Considerations, per this manual. 5. Review the Installation and Startup procedures, per this manual. In most cases, this includes a step-by-step powerup routine. Try it.
	B. If the LEDs are lit, the board may be in the wrong slot.	1. For VMEmodules, the CPU board should be in the first (leftmost) slot. 2. Also check that the "system controller" function on the board is enabled, per this manual.
	C.The "system console" terminal may be configured wrong.	Configure the system console terminal per this manual.

Table F-1. Basic Troubleshooting Steps (Continued)

Condition ...	Possible Problem ...	Try This ...
II. There is a display on the terminal, but input from the keyboard and/or mouse has no effect.	A. The keyboard or mouse may be connected incorrectly.	Recheck the keyboard and/or mouse connections and power.
	B. Board jumpers may be configured incorrectly.	Check the board jumpers per this manual.
	C. You may have invoked flow control by pressing a HOLD or PAUSE key, or by typing **<CTRL>-S** Also, a HOLD LED may be lit.	Press the HOLD or PAUSE key again. If this does not free up the keyboard, type in **<CTRL>-Q**
III. Debug prompt `177-Bug>` does not appear at powerup, and the board does not auto boot.	A. Debugger EPROM/Flash may be missing	1. Disconnect *all* power from your system. 2. Check that the proper debugger EPROM or debugger Flash memory is installed per this manual. 3. Reconnect power. 4. Restart the system by "double-button reset": press the RESET and ABORT switches at the same time; release RESET first, wait seven seconds, then release ABORT. 5. If the debug prompt appears, go to step IV or step V, as indicated. If the debug prompt does not appear, go to step VI.
	B. The board may need to be reset.	

Table F-1. Basic Troubleshooting Steps (Continued)

Condition ...	Possible Problem ...	Try This ...
IV. Debug prompt `177-Bug>` appears at powerup, but the board does not auto boot.	A. The initial debugger environment parameters may be set wrong.	1. Start the onboard calendar clock and timer. Type **set *mmddyyhhmm* <CR>** where the characters indicate the month, day, year, hour, and minute. The date and time will be displayed. **Caution** ! Performing the next step will change some parameters that may affect your system operation. 2. Type in **env;d <CR>** This sets up the default parameters for the debugger environment. 3. When prompted to Update Non-Volatile RAM, type in **y <CR>** 4. When prompted to Reset Local System, type in **y <CR>** 5. After clock speed is displayed, immediately (within five seconds) press the Return key **<CR>** or **BREAK** to exit to System Menu. Then enter a 3 "Go to System Debugger" and Return **3 <CR>** Now the prompt should be `177-Diag>` (continues>)
	B. There may be some fault in the board hardware.	

F

Table F-1. Basic Troubleshooting Steps (Continued)

Condition ...	Possible Problem ...	Try This ...
		6.You may need to use the **cnfg** command (see your board Debugger Manual) to change clock speed and/or Ethernet Address, and then later return to **env <CR>** and step 3. 7. Run selftest by typing in **st <CR>** The tests take as much as 10 minutes, depending on RAM size. They are complete when the prompt returns. (The onboard selftest is a valuable tool in isolating defects.) 8. The system may indicate that it has passed all the selftests. Or, it may indicate a test that failed. If neither happens, enter **de <CR>** Any errors should now be displayed. If there are any errors, go to step VI. If there are no errors, go to step V.
V. The debugger is in system mode and the board auto boots, or the board has passed selftests.	A. No problems - troubleshooting is done.	No further troubleshooting steps are required. **Note** Even if the board passes all tests, it may still be bad. Selftest does not try out all functions in the board (for example, SCSI, or VMEbus tests).
VI. The board has failed one or more of the tests listed above, and can not be corrected using the steps given.	A. There may be some fault in the board hardware or the on-board debugging and diagnostic firmware.	1. Document the problem and return the board for service. 2. Phone 1-800-222-5640.

Index

D

N

O

P

Q

R

S

T

U

V

W

X