MVME167
Single Board Computer
Installation Guide

MVME167IG/D4

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 85282-9602

Preface

This manual provides a general board level hardware description, hardware preparation and installation instructions, debugger general information, and information on using the debugger.

This manual applies to the following MVME167 Single Board Computers:

Assembly Item	Board Description
MVME167-001B	25MHZ, 4MB Parity
MVME167-002B	25MHZ, 8MB Parity
MVME167-003B	25MHZ, 16MB Parity
MVME167-004B	25MHZ, 32MB Parity
MVME167-031B	33MHZ, 4MB ECC
MVME167-032B	33MHZ, 8MB ECC
MVME167-033B	33MHZ, 16MB ECC
MVME167-034B	33MHZ, 32MB ECC
MVME167-035B	33MHZ, 64MB ECC
MVME167-036B	33MHZ, 128MB ECC

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

Anyone using this manual should have a basic knowledge of computers and digital logic.

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.

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

WARNING

This equipment generates, uses, and can radiate electro-magnetic energy. It may cause or be susceptible to electro-magnetic 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
March 1997

Contents

List of Figures

List of Tables

Introduction to the MVME167 Installation Guide 1

This Chapter Covers

- Details about this manual
- Terminology, conventions, and definitions used
- Other publications relevant to the MVME167

About this Manual

This manual supports the setup, installation, and debugging of the CISC-based MVME167 Single Board Computer; a high-functionality VMEbus-based solution for scientific and industrial embedded-controller applications.

This manual provides:

- A general *Board Level Hardware Description* in Chapter 2
- *Hardware Preparation and Installation* instructions in Chapter 3
- *Debugger General Information* in Chapter 4
- Debugger/monitor commands, and other information about *Using the 167Bug Debugger* in Chapter 5
- Other information needed for startup and troubleshooting of the MVME167 CISC Single Board Computer, including
  - *Configure and Environment Commands* in Appendix A
  - *Disk/Tape Controller Data* in Appendix B for controller modules supported by 167Bug
  - *Network Controller Data* in Appendix C
  - Procedures for *Troubleshooting CPU Boards* in Appendix D
  - *EIA-232-D Interconnections* in Appendix E

Terminology, Conventions, and Definitions Used in this Manual

Data and Address Parameter Numeric Formats

Throughout this manual, a character identifying the numeric format precedes data and address parameters as follows:

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

For example, “12” is the decimal number twelve, and “$12” is the decimal number eighteen.

Unless otherwise specified, all address references are in hexadecimal.

Signal Name Conventions

An asterisk (*) follows signal names for signals which are level or edge significant:

Term	* Indicates
level significant	The signal is true or valid when the signal is low.
edge significant	The actions initiated by that signal occur on high to low transition.

Assertion and Negation Conventions

Assertion and negation are used to specify forcing a signal to a particular state. These terms are used independently of the voltage level (high or low) that they represent.

Term	Indicates
Assertion and assert	The signal is active or true.
Negation and negate	The signal is inactive or false.

Data and Address Size Definitions

Data and address sizes are defined as follows:

Name	Size	Numbered	Significance	Called
Byte	8 bits	0 through 7	bit 0 is the least significant	byte
Two-byte	16 bits	0 through 15	bit 0 is the least significant	word
Four-byte	32 bits	0 through 31	bit 0 is the least significant	longword

Control and Status Bit Definitions

The terms control bit and status bit are used extensively in this document to describe certain bits in registers.

Term	Describes
Control bit	The bit can be set and cleared under software control.
Status bit	The bit reflects a specific condition.

- The status bit can be read by software to determine operational or exception conditions.

True/False Bit State Definitions

True and False indicate whether a bit enables or disables the function it controls:

Term	Indicates
True	Enables the function it controls.
False	Disables the function it controls.

Bit Value Descriptions

In all tables, the terms 0 and 1 are used to describe the actual value that should be written to the bit, or the value that it yields when read.

Related Documentation

The MVME167 ships with a startup installation guide (MVME167IG/D, the document you are presently reading) which includes installation instructions, jumper configuration information, memory maps, debugger/monitor commands, and any other information needed for startup of the board.

If you wish to develop your own applications or need more detailed information about your MVME167 Single Board Computer, you may purchase the additional documentation listed on the following pages through your local Motorola sales office.

If any supplements have been issued for a manual or guide, they will be furnished along with the particular document. Each Motorola Computer Group manual publication number is suffixed with characters which represent the revision level of the document, such as "/D2" (the second revision of a manual); a supplement bears the same number as a manual but has a suffix such as "/D2A1" (the first supplement to the second edition of the manual).

Document Set for MVME167-0*xx* Board

You may order the manuals in this list individually or as a set. The manual set 68-M167SET includes:

Motorola Publication Number	Description
MVME167/D	MVME167 Single Board Computer User's Manual
68KBUG1/D 68KBUG2/D	Debugging Package for Motorola 68K CISC CPUs User's Manual (Parts 1 and 2)
MVME167BUG/D	MVME167Bug Debugging Package User's Manual
VMESBCA1/PG VMESBCA2/PG	Single Board Computer Programmer's Reference Guide (Parts 1 and 2)

Motorola Publication Number	Description
SBCSCSI/D	Single Board Computers SCSI Software User's Manual

Additional Manuals for this Board

Also available but not included in the set:

Motorola Publication Number	Description
MVME167IG/D	MVME167 Single Board Computer Installation Guide (this manual).
SIMVME167/D	MVME167 Single Board Computer Support Information. The SIMVME167 manual contains the connector interconnect signal information, parts lists, and the schematics for the MVME167.

Other Applicable Motorola Publications

The following publications are applicable to the MVME167 and may provide additional helpful information. They may be purchased through your local Motorola sales office.

Motorola Publication Number	Description
MVME712M	MVME712M Transition Module and P2 Adapter Board User's Manual
MVME712A	MVME712-12, MVME712-13, MVME712A, MVME712AM, and MVME712B Transition Modules and LCP2 Adapter Board User's Manual

Motorola Publication Number	Description
M68040UM	MC68040 Microprocessors User's Manual

Non-Motorola Peripheral Controllers Publications Bundle

For your convenience, we have collected user's manuals for each of the peripheral controllers used on the MVME167 from the suppliers. This bundle, which can be ordered as part number **68-1X7DS**, includes the following manuals:

Part Number	Description
NCR53C710DM	NCR 53C710 SCSI I/O Processor Data Manual
NCR53C710PG	NCR 53C710 SCSI I/O Processor Programmer's Guide
CL-CD2400/2401	Cirrus Logic CD2401 Serial Controller User's Manual
UM95SCC0100	Zilog Z85230 Serial Communications Controller User's Manual
290218	Intel Networking Components Data Manual
290435	Intel i28F008 Flash Memory Data Sheet
290245	Intel i28F020 Flash Memory Data Sheet
292095	Intel i28F008SA Software Drivers Application Note
292099	Intel i28F008SA Automation and Algorithms Application Note
MK48T08/18B	SGS-THOMSON MK48T08 Time Clock/NVRAM Data Sheet
MC68230/D	MC68230 Parallel Interface Timer (PI/T) Data Sheet
SBCCOMPS/L	Customer Letter for Component Alternatives

Applicable Non-Motorola Publications

The following non-Motorola publications are also available from the sources indicated.

Document Title	Source
Versatile Backplane Bus: VMEbus, ANSI/IEEE Std 1014-1987 (VMEbus Specification) (This is also *Microprocessor System Bus for 1 to 4 Byte Data,* IEC 821 BUS)	The Institute of Electrical and Electronics Engineers, Inc. 345 East 47th St. New York, NY 10017
	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 15 Inverness Way East Englewood, CO 80112-5704
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	NCR Corporation Microelectronics Products Division 1635 Aeroplaza Dr. Colorado Springs, CO 80916
NCR 53C710 SCSI I/O Processor Programmer's Guide, order number NCR53C710PG	
MK48T08(B) Timekeeper ™ and 8Kx8 Zeropower ™ RAM data sheet in Static RAMs Databook, order number DBSRAM71	SGS-THOMSON Microelectronics Group Marketing Headquarters 1000 East Bell Rd. Phoenix, AZ 85022-2699

Board Level Hardware Description 2

This Chapter Covers

- A general description of the MVME167 CISC Single Board Computer
- Features and specifications
- A board-level hardware overview
- A detailed hardware functional description, including front panel switches and indicators
- Memory maps

General Description

The MVME167, based on the MC68040 microprocessor, is a high-functionality VMEbus-based solution for scientific and industrial embedded-controller applications. It features:

- Onboard memory expansion mezzanine module with 4, 8, 16, 32, 64, or 128 MB of onboard DRAM
- SCSI bus interface with DMA
- Four serial ports with EIA-232-D interface
- Centronics (parallel) printer port
- Ethernet transceiver interface with DMA
- 167Bug debug monitor firmware

Onboard Memory Mezzanine Module

The MVME167 onboard DRAM mezzanine boards are available in different sizes and with programmable parity protection or Error Checking and Correction (ECC) protection.

- The main board and a single mezzanine board together take one slot.
- Motorola software supports mixed parity and ECC memory boards on the same main board.
- Mezzanine board sizes are 4, 8, 16, or 32MB (parity), or 4, 8, 16, 32, 64, or 128MB (ECC),
  - Two mezzanine boards may be stacked to provide 256MB of onboard RAM (ECC) or 64 MB (parity). The stacked configuration requires two VMEbus slots.
- The DRAM is four-way interleaved to efficiently support cache burst cycles.
- The parity mezzanines are only supported on 25 MHz main boards.

A functional description of the Onboard DRAM starts on page 2-15.

SCSI Mass Storage Interface

The MVME167 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

A functional description of the SCSI Interface starts on page 2-22.

Serial Ports

The serial ports support standard baud rates of 110 to 38.4K baud.

Serial Port	Function	Synchronous/ Asynchronous	Signals	Bit Rates
1	Minimum	Asynchronous	RXD, CTS, TXD, and RTS	
2 and 3	Full	Asynchronous	RXD, CTS, DCD, TXD, RTS, and DTR	
4	Full	Both	RXD, CTS, DCD, TXD, RTS, and DTR	Synchronous up to 64 k bits per second

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.

A functional description of the Serial Port Interface starts on page 2-18.

Parallel (Printer) Port

The 8-bit bidirectional parallel port may be used as a Centronics-compatible parallel printer port or as a general parallel I/O port.

A functional description of the Printer Port interface starts on page 2-20.

Ethernet Transceiver Interface

The Ethernet transceiver interface is located on the MVME167, and the industry standard connector is located on the MVME712X transition module.

A functional description of the Ethernet Interface starts on page 2-21.

167Bug Firmware

The MVME167Bug debug monitor firmware (167Bug) is provided in two of the four EPROM sockets on the MVME167.

It provides:

- Over 50 debug commands
- Up/down load commands
- Disk bootstrap load commands
- A full set of onboard diagnostics
- A one-line assembler/disassembler

The 167Bug command-line interface accepts commands from the system console terminal.

167Bug can also operate in a System Mode, which includes choices from a service menu.

Features

- MC68040 Microprocessor
- 4/8/16/32/64MB of 32-bit DRAM with parity protection or 4/8/16/32/64/128/256MB of 32-bit DRAM with ECC protection
- Four 44-pin PLCC ROM sockets (organized as two banks of 32 bits)
- 128KB Static RAM (with optional battery backup as a factory build special request)
- Status LEDs for FAIL, STAT, RUN, SCON, LAN, +12V (LAN power), SCSI, and VME.
- 8K by 8 static RAM and time-of-day clock with battery backup

- RESET and ABORT switches
- Four 32-bit tick timers for periodic interrupts
- Watchdog timer
- Eight software interrupts
- I/O
  - SCSI Bus interface with DMA
  - Four serial ports with EIA-232-D buffers with DMA
  - Centronics printer port
  - Ethernet transceiver interface with DMA
- VMEbus interface
  - VMEbus system controller functions
  - VMEbus interface to local bus (A24/A32, D8/D16/D32 and D8/D16/D32/D64BLT) (BLT = Block Transfer)
  - Local bus to VMEbus interface (A16/A24/A32, D8/D16/D32)
  - VMEbus interrupter
  - VMEbus interrupt handler
  - Global CSR for interprocessor communications
  - DMA for fast local memory - VMEbus transfers (A16/A24/A32, D16/D32 and D16/D32/D64BLT)

2

Specifications

Table 2-1. MVME167 General Specifications

Characteristics		Specifications	
Power requirements (with all four EPROM sockets populated and excluding external LAN transceiver)		+5 Vdc (+/- 5%)	3.5 A (typical), 4.5 A (max.) (at 25 MHz, with 32MB parity DRAM)
			5.0 A (typical), 6.5 A (max.) (at 33 MHz, with 128MB ECC DRAM)
		+12 Vdc (+/- 5%)	100 mA (maximum) (1.0 A (max.) with offboard LAN transceiver)
		-12 Vdc (+/- 5%)	100 mA (maximum)
Operating temperature		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 Double-high VMEboard	PC board with mezzanine module only	Height	9.187 inches (233.35 mm)
		Depth	6.299 inches (160.00 mm)
		Thickness	0.662 inches (16.77 mm)
	PC board with connectors and front panel	Height	10.309 inches (261.85 mm)
		Depth	7.4 inches (188 mm)
		Thickness	0.80 inches (20.32 mm)

Conformance to Requirements

These boards are designed to conform to the requirements of the following specifications:

- VMEBus Specification (IEEE 1014-87)
- EIA-232-D Serial Interface Specification, EIA
- SCSI Specification

Board Level Overview

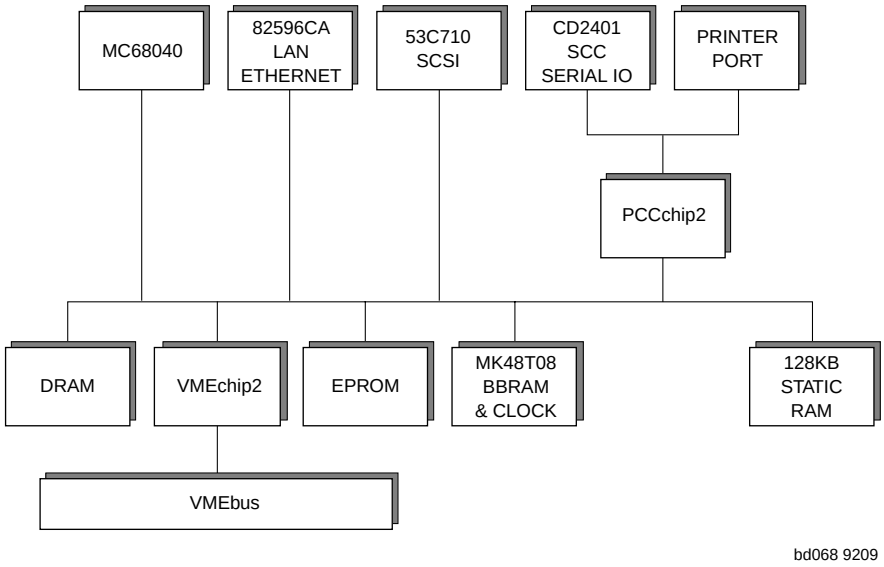


Figure 2-1. MVME167 General Block Diagram

Connectors

The MVME167 has two 96-position DIN connectors: P1 and P2.

- P1 rows A, B, C, and P2 row B provide the VMEbus interconnection.
- P2 rows A and C provide the connection to the SCSI bus, serial ports, Ethernet, and printer.

Adapters

I/O on the MVME167 is connected to the VMEbus P2 connector.

The main board is connected to the transition modules through a P2 adapter board and cables.

Transition Modules

The MVME712X transition modules provide configuration headers and provide industry standard connectors for the I/O devices. Refer to Figure 3-3 on page 3-21.

- The MVME167 supports the transition modules MVME712-12, MVME712-13, MVME712M, MVME712A, MVME712AM, and MVME712B (referred to in this manual as MVME712X, unless separately specified).

Transition modules and adapter boards are covered in the *MVME712M Transition Module and P2 Adapter Board User's Manual*, and the *MVME712-12, MVME712-13, MVME712A, MVME712AM, and MVME712B Transition Modules and LCP2 Adapter Board User's Manual.*

ASICs

The MVME167 board features several Application Specific Integrated Circuits (ASICs) including:

- VMEchip2
- PCCchip2
- MEMC040
- MCECC

All programmable registers in the MVME167 that reside in ASICs are covered in the *Single Board Computers Programmer's Reference Guide*.

VMEchip2 ASIC

Provides the VMEbus interface. The VMEchip2 includes:

- Two tick timers
- A watchdog timer
- Programmable map decoders for the master and slave interfaces, and a VMEbus to/from local bus DMA controller
- A VMEbus to/from local bus non-DMA programmed access interface
- A VMEbus interrupter
- A VMEbus system controller
- A VMEbus interrupt handler
- A VMEbus requester.

Table 2-2. Bus Transfers

Transfer type	Can be...
Processor-to-VMEbus	D8, D16, or D32
VMEchip2 DMA to the VMEbus	D16, D32, D16/BLT, D32/BLT, or D64/MBLT

PCCchip2 ASIC

The PCCchip2 ASIC provides two tick timers and the interface to the:

- LAN chip
- SCSI chip
- Serial port chip
- Printer port
- BBRAM

MEMC040 Memory Controller ASIC

The MEMC040 memory controller ASIC provides the programmable interface for the parity-protected DRAM mezzanine board.

MCECC Memory Controller ASIC

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

Functional Description

The major functional blocks of the MVME167 covered in this section are:

- Front panel switches and LED indicators
- Data bus structure
- MC68040 CPU
- EPROM
- SRAM
- Onboard DRAM
- Battery backed up RAM and clock
- VMEbus interface
- I/O interfaces
- Local resources

Front Panel Switches and LEDs

There are two switches and eight LEDs on the board's front panel (refer to Table 2-3, Table 2-4, and Figure 3-1 on page 3-6).

Table 2-3. Front Panel Switches

Switch Name	Description
RESET	The RESET switch resets all onboard devices and drives SYSRESET* if the board is system controller. The RESET switch may be disabled by software.
ABORT	When enabled by software, the ABORT switch generates an interrupt at a user-programmable level. It is normally used to abort program execution and return to the debugger.

Table 2-4. Front Panel LEDs

LED Name	Color	Description
FAIL	Red	The FAIL LED lights when the BRDFAIL signal line is active.
STAT	Yellow	The STAT LED lights when the MC68040 is halted.
RUN	Green	The RUN LED lights when the local bus TIP* signal line is low. This indicates one of the local bus masters is executing a local bus cycle.
SCON	Green	The SCON LED lights when the MVME167 is the VMEbus system controller.
LAN	Green	The LAN LED lights when the LAN chip is local bus master.
+12V	Green	The +12V LED lights when +12V power is available to the Ethernet transceiver interface.
SCSI	Green	The SCSI LED lights when the SCSI chip is local bus master.
VME	Green	The VME LED 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).

Data Bus Structure

The local data bus on the MVME167 is a 32-bit synchronous bus that is based on the MC68040 bus, and supports burst transfers and snooping.

Local Bus Arbitration

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:

1. 82596CA LAN (highest)
2. CD2401 serial (through the PCCchip2)
3. 53C710 SCSI
4. VMEbus
5. MPU (lowest)

In general, 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 its port size, data bus connection, and any restrictions that apply when accessing the device.

MC68040 MPU

The MC68040 processor is used on the MVME167. The MC68040 has onchip instruction and data caches and a floating point processor. Refer to the M68040 user's manual for more information.

EPROM

Four 44-pin PLCC/CLCC EPROM sockets for 27C102JK or 27C202JK type EPROMs. They are:

- Organized as two 32-bit wide banks that support 8-, 16-, and 32-bit read accesses
- Controlled by the VMEchip2
- Mapped to local bus address 0 following a local bus reset
  - This allows the MC68040 to access the stack pointer and execution address following a reset

Programmable EPROM features

- Map decoder
- Access time
- When accessible at address 0

Static RAM

The MVME167 includes 128KB of 32-bit wide 100 ns static RAM (SRAM), which:

- Supports 8-, 16-, and 32-bit wide accesses
- Allows the debugger operation and execution of limited diagnostics without the DRAM mezzanine
- Is controlled by the VMEchip2; the access time is programmable.

Optional SRAM Battery Backup

SRAM battery backup is optionally available on the MVME167, *but only as a factory build and only by special request*. (Contact your local Motorola sales office for details.) The battery backup function is provided by a Dallas DS1210S nonvolatile controller chip and a Sanyo CR2430 battery.

The onboard power source is a Sanyo CR2430 battery which is socketed for easy removal and replacement. A small capacitor is provided to allow the battery to be quickly replaced without data loss (i.e., the battery must be replaced within 30 seconds).

If your MVME167 is equipped with SRAM battery backup; when the main board power fails, the DS1210S selects the battery as the power source.

Each time the board is powered, the DS1210S checks the power source, allowing software to provide an early warning to avoid data loss:

- If the voltage of the backup source is less than two volts, the second memory access cycle is blocked.
- Because the DS1210S may block the second access, the software should do at least two accesses before relying on the data.

With the optional battery backup, the MVME167 provides jumpers (on *Optional SRAM Backup Power Source Select Header J8* on page 3-12) that allow the power source of the DS1210S to be connected to the VMEbus +5 V STDBY pin or to the onboard battery.

Caution

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.
- Do not charge.
- Always check proper polarity.

To remove the battery from the module, carefully pull the battery from the socket. (Data will be lost if a new battery is not installed within 30 seconds.)

Before installing a new battery, ensure that the battery pins are clean. Note the battery polarity and press the battery into the socket.

Onboard DRAM

The MVME167 onboard DRAM is located on a mezzanine board. The mezzanine boards are available in different sizes and with parity protection or ECC protection.

Note Parity mezzanines are only supported on 25MHz main boards.

Motorola software *does* support mixed parity and ECC memory boards on the same main board.

The DRAM is four-way interleaved to efficiently support cache burst cycles.

Onboard DRAM mezzanines are available in these configurations:

- 4, 8, 16, or 32MB with parity protection
- 4, 8, 16, 32, 64, or 128MB with ECC protection

Stacking Mezzanines

Two mezzanine boards may be stacked to provide up to 256MB of onboard RAM (ECC).

- The main board and a single mezzanine board together take one slot.
- The stacked configuration requires two VMEboard slots.

DRAM Programming Considerations

- 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.
- 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 installation. However, software should insure a minimum of 10 initialization cycles are performed to each bank of RAM.

Refer to the MEMC040 or the MCECC in the *Single Board Computers Programmer's Reference Guide* for detailed programming information.

Battery Backed Up RAM and Clock

The MK48T08 RAM and clock chip is a 28-pin package that provides:

- A time-of-day clock
- An oscillator
- A crystal
- Power fail detection
- Memory write protection
- 8KB of RAM
- A battery

The clock provides

- Seconds, minutes, hours, day, date, month, and year in BCD 24-hour format
- Automatic corrections for 28-, 29- (leap year), and 30-day months

No interrupts are generated by the clock.

The MK48T08 is an 8 bit device; however, the interface provided by the PCCchip2 supports 8-, 16-, and 32-bit accesses to the MK48T08.

Refer to the MK48T08 data sheet for detailed programming information.

VMEbus Interface

The VMEchip2 provides:

- Local bus to VMEbus interface
- VMEbus to local bus interface
- Local-VMEbus DMA controller functions
- VMEbus system controller functions

I/O Interfaces

The MVME167 provides onboard I/O for many system applications.

- The I/O functions include:
  - Serial ports
  - Printer port
  - Ethernet transceiver interface
  - SCSI mass storage interface.
- An external I/O transition module such as the MVME712X should be used to convert the I/O connector pinout to industry-standard connectors.
- The I/O interface configuration headers are located on the MVME167 and the MVME712X transition module.

The I/O on the MVME167 is connected to the VMEbus P2 connector. The MVME712X transition module is connected to the MVME167 through cables and a P2 adapter board.

Serial Port Interface

The CD2401 serial controller chip (SCC) implements 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.

All four serial ports use EIA-232-D drivers and receivers located on the MVME167, and all the signal lines are routed to the I/O connector.

- Serial port 1 is a minimum function asynchronous port. It uses RXD, CTS, TXD, and RTS.
- Serial ports 2 and 3 are full function asynchronous ports. They use RXD, CTS, DCD, TXD, RTS, and 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, and DTR. It also interfaces to the synchronous clock signal lines.

Serial Interface Programming Considerations

- The MVME167 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
- Accesses to the CD2401 must be 8 or 16 bits: 32-bit accesses are not permitted.
- 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.

Refer to the CD2401 data sheet for detailed programming information.

Parallel Port Interface

The PCCchip2 provides an 8-bit bidirectional parallel port. This port may be used as a Centronics-compatible parallel printer port or as a general parallel I/O port.

All eight bits of the port must be either inputs or outputs (no individual bit selection).

In addition to the 8 bits of data, there are two control pins and five status pins.

When used as a parallel printer port, these pins function as follows:

Status Pins	Printer Acknowledge (ACK*)
	Printer Fault (FAULT*)
	Printer Busy (BSY)
	Printer Select (SELECT)
	Printer Paper Error (PE)
Control Pins	Printer Strobe (STROBE*)
	Input Prime (INP*)

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

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.

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.

The Ethernet transceiver interface is located on the MVME167, and the industry-standard connector is located on the MVME712X transition module.

Every MVME167 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 MVME167 has a different value for *xxxxx*).

Each module has the 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.

The MVME167 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.

Buffer Overruns

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. Support functions for the 82596CA are provided by the PCCchip2.

Refer to the 82596CA user's guide for detailed programming information.

SCSI Interface

The MVME167 provides for mass storage subsystems through the industry-standard SCSI bus. These subsystems may include hard and floppy disk drives, streaming tape drives, and other mass storage devices.

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

Support functions for the 53C710 are provided by the PCCchip2. Refer to the 53C710 user's guide for detailed programming information.

SCSI Termination

Because this board has no provision for SCSI termination, you must ensure that the SCSI bus is terminated properly.

- If the SCSI bus ends at the P2 adapter board, then termination resistors must be installed on the P2 adapter board. Note: +5V power to the SCSI bus TERM power line and termination resistors is provided through a fuse located on the P2 adapter board.
- If there are additional SCSI mass storage devices in your system ensure that terminators are installed on the last device in the SCSI chain.

Local Resources

The MVME167 includes many resources for the local processor. These include tick timers, software programmable hardware interrupts, watchdog timer, and local bus timeout.

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.

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.

Software-Programmable Hardware Interrupts

Eight software-programmable hardware interrupts are provided by the VMEchip2. These interrupts allow software to create a hardware interrupt.

Local Bus Timeout

The MVME167 provides a timeout 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 timeout value is selectable by software for 8 µsec, 64 µsec, 256 µsec, or 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.

2

Memory Maps

There are two points of view for memory maps:

1. Local bus memory map
   - The mapping of all resources as viewed by local bus masters
2. VMEbus memory map
   - The mapping of onboard resources as viewed by VMEbus masters

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.

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 MVME167, Transfer Types 0, 1, and 2 define the normal address range.

Table 2-5 on page 2-25 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 2-6 on page 2-26 further defines the map for the local I/O devices.

Table 2-5. Local Bus Memory Map

Address Range	Devices Accessed	Port Size	Size	Software Cache Inhibit	Notes
$00000000 - DRAMSIZE	User Programmable (Onboard DRAM)	**D32**	DRAMSIZE	N	1, 2
DRAMSIZE - $FF7FFFFF	User Programmable (VMEbus)	**D32/D16**	3GB	?	3, 4
$FF800000 - $FFBFFFFF	ROM	**D32**	4MB	N	1
$FFC00000 - $FFDFFFFF	reserved	**--**	2MB	--	5
$FFE00000 - $FFE1FFFF	SRAM	**D32**	128KB	N	--
$FFE20000 - $FFEFFFFF	SRAM (repeated)	**D32**	896KB	N	--
$FFF00000 - $FFFEFFFF	Local I/O Devices (Refer to next table)	**D32-D8**	1MB	Y	3
$FFFF0000 - $FFFFFFFF	User Programmable (VMEbus A16)	**D32/D16**	64KB	?	2, 4

Notes

1. Onboard EPROM appears at $00000000 - $003FFFFF following a local bus reset. The EPROM appears at 0 until the ROM0 bit is cleared in the VMEchip2. The ROM0 bit is located at address $FFF40030 bit 20. The EPROM must be disabled at 0 before the DRAM is enabled. The VMEchip2 and DRAM map decoders are disabled by a local bus reset.
2. This area is user-programmable. The suggested use is shown in the table. The DRAM decoder is programmed in the MEMC040 or 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.

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

Table 2-6. 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	1,3
$FFF40100 - $FFF401FF	VMEchip2 (GCSR)	D32-D8	256B	1,3
$FFF40200 - $FFF40FFF	reserved	--	3.5KB	4,6
$FFF41000 - $FFF41FFF	reserved	--	4KB	4
$FFF42000 - $FFF42FFF	PCCchip2	D32-D8	4KB	1
$FFF43000 - $FFF430FF	MEMC040/MCECC #1	D8	256B	1
$FFF43100 - $FFF431FF	MEMC040/MCECC #2	D8	256B	1
$FFF43200 - $FFF43FFF	MEMC040s/MCECCs (repeated)	--	3.5KB	1,6
$FFF44000 - $FFF44FFF	reserved	--	4KB	4
$FFF45000 - $FFF451FF	CD2401 (Serial Comm. Cont.)	D16-D8	512B	1
$FFF45200 - $FFF45DFF	reserved	--	3KB	6,8
$FFF45E00 - $FFF45FFF	reserved	--	512B	8
$FFF46000 - $FFF46FFF	82596CA (LAN)	D32	4KB	1,7
$FFF47000 - $FFF47FFF	53C710 (SCSI)	D32/D8	4KB	1
$FFF48000 - $FFF4FFFF	reserved	--	32KB	4
$FFF50000 - $FFF6FFFF	reserved	--	128KB	4
$FFF70000 - $FFF76FFF	reserved	--	28KB	5
$FFF77000 - $FFF77FFF	reserved	--	4KB	2
$FFF78000 - $FFF7EFFF	reserved	--	28KB	5
$FFF7F000 - $FFF7FFFF	reserved	--	4KB	2
$FFF80000 - $FFF9FFFF	reserved	--	128KB	5

Table 2-6. Local I/O Devices Memory Map (Continued)

Address Range	Devices Accessed	Port Size	Size	Notes
$FFFA0000 - $FFFBFFFF	reserved	--	128KB	4
$FFFC0000 - $FFFCFFFF	MK48T08 (BBRAM, TOD Clock)	D32-D8	64KB	1
$FFFD0000 - $FFFDFFFF	reserved	--	64KB	4
$FFFE0000 - $FFFEFFFF	reserved	--	64KB	2

Notes

1. For a complete description of the register bits, refer to the *Single Board Computers Programmer's Reference Guide* or to the data sheet for the specific chip.
2. On the MVME167 this area does not return an acknowledge signal. If the local bus timer on the MVME167 is enabled, the access times out and is terminated by a TEA signal.
3. 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.
4. This area does not return an acknowledge signal. If the local bus timer is enabled, the access times out and is terminated by a TEA signal.
5. This area does return an acknowledge signal.
6. Size is approximate.
7. Port commands to the 82596CA must be written as two 16-bit writes: upper word first and lower word second.
8. The CD2401 appears repeatedly from $FFF45200 to $FFF45FFF. If the local bus timer is enabled, the access times out and is terminated by a TEA signal.

VMEbus Memory Map

This section describes the mapping of local resources as viewed by VMEbus masters. Default addresses for the slave, master, and GCSR address decoders are provided by the **ENV** command. Refer to Appendix A.

VMEbus Accesses to the Local Bus

The VMEchip2 includes a user-p87grammable map decoder for the VMEbus to local bus interface. The map decoder allows you to program the starting and ending address and the modifiers the MVME167 responds to.

VMEbus Short I/O Memory Map

The VMEchip2 includes a user-programmable map decoder for the GCSR. The GCSR map decoder allows you to program the starting address of the GCSR in the VMEbus short I/O space.

Hardware Preparation and Installation 3

This Chapter Covers

This chapter provides instructions on:

- Unpacking the equipment
- Preparing the hardware
- Installing the MVME167 CISC Single Board Computer

Note that hardware preparation instructions for the MVME712X transition module are provided in separate user's manuals for each model. Refer to the user's manual you received with your MVME712X.

Unpacking the Equipment

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

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

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

Overview of Startup 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 chapter and all Caution notes, and have the related documentation with you before you begin.

Table 3-1. Startup Overview

Stage	What you will need to do...		Refer to...	On page...
1	**Prepare the MVME167.**		*Preparing the Hardware*	3-5
		Ensure that EPROM devices are properly installed in their sockets.	*Checking the 167Bug EPROMs*	3-7
		Configure adapters and MVME712X transition modules.	The user's manual you received with your MVME712X module	
		Install/remove jumpers on headers.	*Jumper Settings*	3-7
2	**Prepare the chassis.**		*Preparing the MVME167 for Installation*	3-13
		Turn power off to chassis and peripherals.	The user's manual you received with your chassis	3-13
		Disconnect AC power cable.		
		Remove chassis cover.		
		Remove filler panels from card slots.		
3	**Install your MVME167 in the chassis.**		*Installing Transition Modules and Adapter Boards*	3-15
		Remove IACK and BG jumpers from backplane.		
		Slide the module into the chassis and fasten it securely.		

Table 3-1. Startup Overview (Continued)

Stage	What you will need to do...	Refer to...	On page...
4	**Install adapter boards and transition modules.**	*Transition Modules and Adapter Boards Overview*	3-16
		Installing Transition Modules and Adapter Boards	3-19
	Set jumpers on the transition module(s).	The user's manual you received with your MVME712X	
	Connect and install the MVME712X transition module.		
	Connect and install the P2 adapter board.		
5	**Connect peripherals.**	*Connecting Peripherals*	3-19
	Connect and install any optional SCSI device cables.	You may also wish to obtain the ***Single Board Computer SCSI Software User's Manual***	
	Connect a console terminal to the MVME712X.		
	Connect any other optional devices or equipment you will be using, such as serial or parallel printers, host computers, etc.	*EIA-232-D Interconnections*	E-1
		Port Numbers	5-9
		Disk/Tape Controller Data	B-1
6	**Complete the installation.**	The user's manual you received with your chassis	3-23
	Reassemble chassis.		
	Reconnect AC power.		

Table 3-1. Startup Overview (Continued)

Stage	What you will need to do...		Refer to...	On page...
7	**Start up the system.**		*Starting the System*	3-23
		Power up the system.	*Front Panel Switches and LEDs*	2-11
		Initialize the real-time clock.	*Initializing the Real-Time Clock*	3-24
		Note that the debugger prompt appears.	*Powering Up the System*	3-24
			Starting Up 167Bug	4-6
			You may also wish to obtain the ***Debugging Package for Motorola 68k CISC CPUs User's Manual*** and the ***167Bug Diagnostics User's Manual***	
		Examine and/or change environmental parameters.	*Examining and/or Changing Environmental Parameters*	3-24
			Setting Environment to Bug/Operating System	A-3
		Program the PCCchip2 and VMEchip2.	*Memory Maps*	2-24
	Troubleshoot the system.		*Troubleshooting the MVME167: Solving Startup Problems*	D-1
		Solve any startup problems.		

Preparing the Hardware

This section covers:

- Modifying hardware configurations before installation
- Checking the 167Bug EPROMs
- Factory jumper settings
- Preparing your MVME167
- Preparing the system chassis

Modifying Configuration before Installation

To select the desired configuration and ensure proper operation of the MVME167, certain option modifications may be necessary before installation.

The location of the switches, jumper headers, connectors, and LED indicators on the MVME167 is illustrated in Figure 3-1.

Option Modification

The MVME167 has provisions for option modification via:

- Software control for most options
- Jumper settings on headers for some options
- Bit settings in control registers after installation for most other options
  - Control registers are described in the *Single Board Computer Programmer's Reference Guide* as listed in *Related Documentation* in Chapter 1 of this manual.

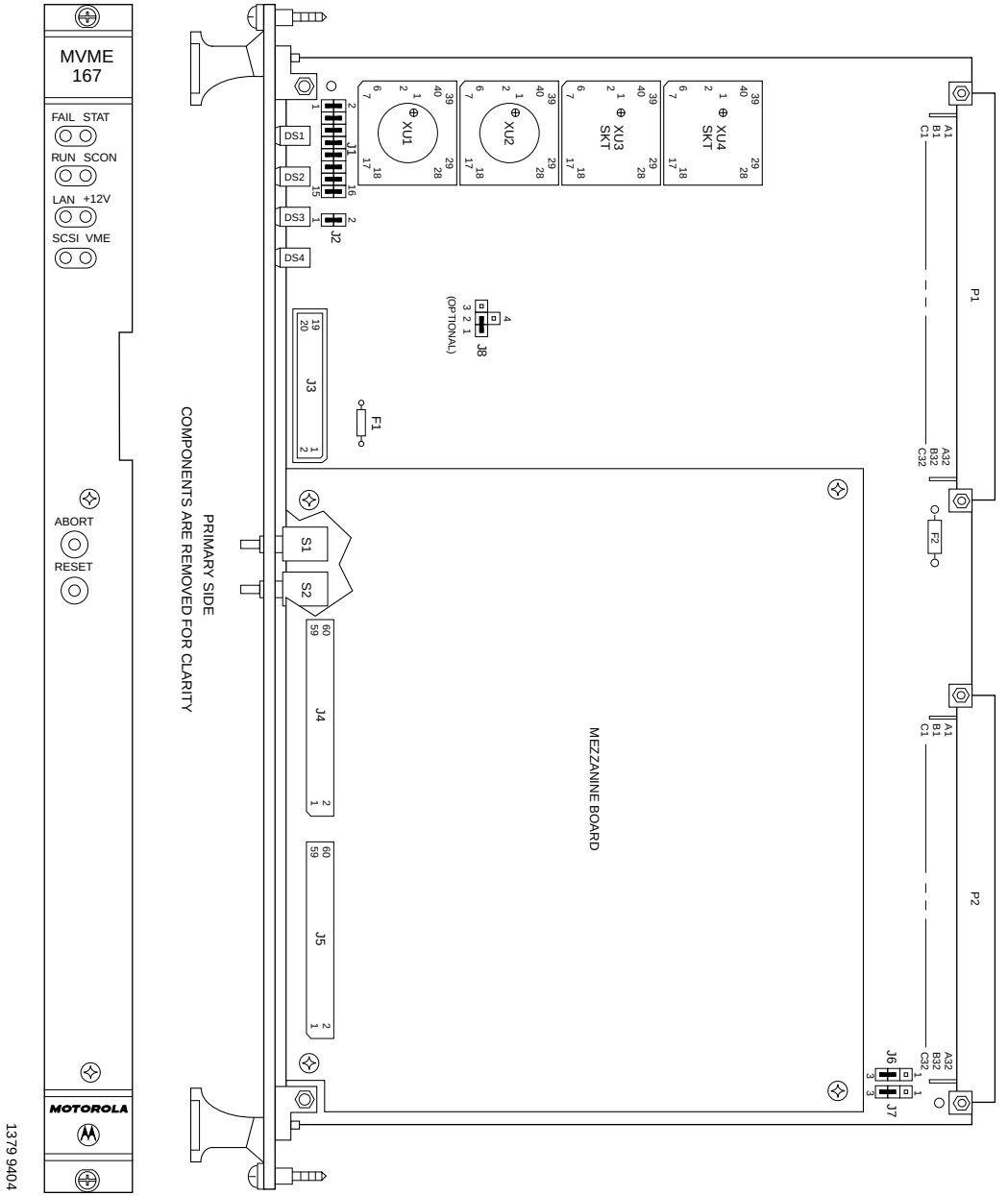


Figure 3-1. MVME167 Switches, Headers, Connectors, Fuses, and LEDs

Checking the 167Bug EPROMs

Be sure that the two factory installed 128K x 16 167Bug EPROMs are in the proper sockets.

EPROM Location

- Odd-numbered label (such as B01): EPROM in socket XU1 (for Least Significant Words)
- Even-numbered label (such as B02): EPROM in XU2 (for Most Significant Words)

EPROM Orientation

Be sure that physical chip orientation is correct:

- The flatted corner of each EPROM aligns with the corresponding portion of the EPROM socket on the MVME167.

User-programmed EPROMs

There are two spare EPROM sockets, XU3 and XU4, available to carry user-programmed EPROMs.

Jumper Settings

The MVME167 has been factory tested and is shipped with the factory jumper settings described in the following sections. The MVME167 operates with its required and factory-installed Debug Monitor, 167Bug, with these factory jumper settings.

Optional Jumper Settings

Most of the optional functions on your board can be changed through software control or bit settings in control registers. If your installation requires it, however, you may change jumper settings on the following headers:

- Jumper pins 9 through 16 on header **J1** are general purpose software readable jumpers open to your application.
- Header **J2** enables/disables the MVME167 as system controller.
- Headers **J6** and **J7** select serial port 4 to drive or receive TRXC4 and RTXC4 clock signals.
- Optional header **J8** selects the SRAM backup power source (this is only available as an optional factory build special request).

General Purpose Software Readable Header J1

Each MVME167 may be configured with readable jumpers. They 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.

Reserved/Defined Bits

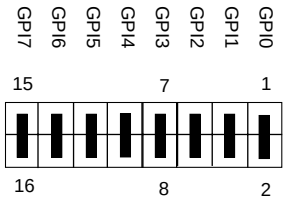
Jumpers on header J1 affect 167Bug operation as listed in Table 3-2. The factory (default) configuration is with all eight jumpers installed (see Table 3-3).

The MVME167Bug reserves/defines the four lower order bits (GPI3 to GPI0). Table 3-2 describes the bits reserved/defined by the debugger:

Table 3-2. J1 Bit Descriptions

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.). This bit will be high when jumper is removed.
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 A) for the ROM defaults. This bit will be high when jumper is removed.
Bit #2 (GPI2)	5-6	Reserved for future use.
Bit #3 (GPI3)	7-8	Reserved for future 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.

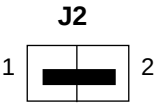
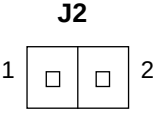
Table 3-3. Factory Settings for J1 General Purpose Readable Jumpers

Header Number	Header Description	Configuration	Jumpers
J1	General purpose software readable jumpers	GPI0 - GPI3: Reserved GPI4 - GPI7: User-definable (Factory configuration)	

System Controller Header J2

The MVME167 can be VMEbus system controller. The system controller function is enabled by installing a jumper on header J2 (see Table 3-4). When the MVME167 is system controller, the SCON LED is turned on.

Table 3-4. Settings for J2 System Controller Header

Header Number	Header Description	Configuration	Jumpers
J2	System controller header	System controller (Factory configuration)	
		Not system controller	

Serial Port 4 Clock Configuration Select Headers J6 and J7

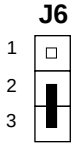
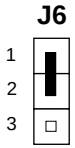
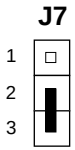
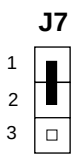
Serial port 4 can be configured to use clock signals provided by the RTXC4 and TRXC4 signal lines.

Headers J6 and J7 on the MVME167 configure serial port 4 to drive or receive RTXC4 and TRXC4, respectively (see Table 3-5).

- Factory configuration is with port 4 set to receive both signals.
- The alternative configuration sets port 4 to drive both signals.

The remaining configuration of the clock lines is accomplished using the Serial Port 4 Clock Configuration Select header on the MVME712M transition module. Refer to the *MVME712M Transition Module and P2 Adapter Board User's Manual* for configuration of that header.

Table 3-5. Settings for J6 and J7 Serial Port 4 Clock Configuration Select Headers

Header Number	Header Description	Configuration	Jumpers
J6	Serial Port 4 clock configuration select headers	Receive RTXC4 (Factory configuration)	
		Drive RTXC4	
J7		Receive TRXC4 (Factory configuration)	
		Drive TRXC4	

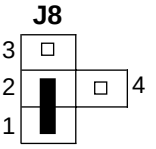
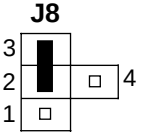
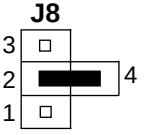
Optional SRAM Backup Power Source Select Header J8

Header J8 is an optional header used to select the SRAM backup power source on the MVME167, if the optional battery is present. (The battery backup for SRAM is optionally available, *but only as a factory build and only by special request*.)

Caution

If your system is equipped with the optional battery backup, do not remove the jumper from J8. This will disable the SRAM. If your MVME167 contains optional header J8, but the optional battery is removed, the jumper must be installed between pins 2 and 4 to disable the backup or between pins 1 and 2 for the factory configuration as shown in Table 3-6.

Table 3-6. Settings for Optional J8 SRAM Backup Power Source Select Header

Header Number	Header Description	Configuration	Jumpers
J8	SRAM backup power source select header	VMEbus +5V STBY (Factory configuration)	
		Optional battery	
		Backup power disabled	

Preparing the MVME167 for Installation

Refer to the setup procedures in the manuals for your particular chassis or system for additional details concerning the installation of the MVME167 into a VME chassis.

Table 3-7. MVME167 Preparation Procedure

Step	Action
1	Install/remove jumpers on headers according to the ***Jumper Settings*** in this chapter and as required for your particular application. ❑ Jumpers on header J1 affect 167Bug operation as listed in ***Jumper Settings***. The default condition is with all eight jumpers installed. ❑ A jumper installed/removed on header J2 enables/disables the system controller function of the MVME167. ❑ Install jumpers on headers J6 and J7 to configure serial port 4 to use clock signals provided by the TRXC4 and RTXC4 signal lines.
2	Configure adapters and transition modules according to the *MVME712X transition module* user's manuals.
3	Ensure that EPROM devices are properly installed in their sockets. ❑ Factory configuration is with two EPROMs installed for the MVME167Bug debug monitor, in sockets XU1 and XU2.

Preparing the System Chassis

Now that the MVME167 is ready for installation, prepare the system chassis and determine slot assignments (for peripherals, transition modules, etc.) as follows:

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.

Table 3-8. Chassis Preparation/Slot Selection Procedure

Step	Action	
1	Turn all power OFF to chassis and peripherals.	
2	Disconnect AC power cable from source.	
3	Remove chassis cover as shown in the user's manual for your particular chassis or system.	
4	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).	
	If the MVME167 is configured as system controller: It must be installed in the left-most card slot (slot 1) to correctly initiate the bus-grant daisy-chain and to have proper operation of the IACK-daisy-chain driver.	**If it is not configured as system controller:** It may be installed in any double-height unused card slot.

Installing the Hardware

This section covers

- Installation of the MVME167 into a VME chassis
- Overview and installation of transition modules and adapter boards
- Connection of peripheral equipment such as console terminals, optional SCSI drives, and serial or parallel printers

Installing the MVME167 in the Chassis

Note that if the MVME167 is to be used as system controller, it must installed in the left-most card slot (slot 1), otherwise it may be installed in any unused double-height card slot.

Table 3-9. MVME167 Installation Procedure

Step	Action
1	Remove IACK and BG jumpers from backplane for the card slot that the MVME167 is to be installed in.
2	Carefully slide the MVME167 into the card slot in the **front** of the chassis. - The MVME167 requires power from both P1 and P2. 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 MVME167 in the chassis with screws provided, making good contact with the transverse mounting rails to minimize RFI emissions

Transition Modules and Adapter Boards Overview

The MVME167 supports the MVME712-12, MVME712-13, MVME712M, MVME712A, MVME712AM, and MVME712B transition modules (referred to in this manual as MVME712X, unless separately specified).

Note Other modules in the system may have to be moved to allow space for the MVME712M which has a double-wide front panel.

MVME712X transition modules provide configuration headers and industry-standard connectors for internal and external I/O devices. The I/O on the MVME167 is connected to the VMEbus P2 connector.

- The MVME712X transition module is connected to the MVME167 through cables and a P2 adapter board as shown in Figure 3-2 on page 3-17.

Note Some cable(s) are not provided with the MVME712X module(s), and therefore must be made or provided by the user. (Motorola recommends using shielded cables for all connections to peripherals to minimize radiation.)

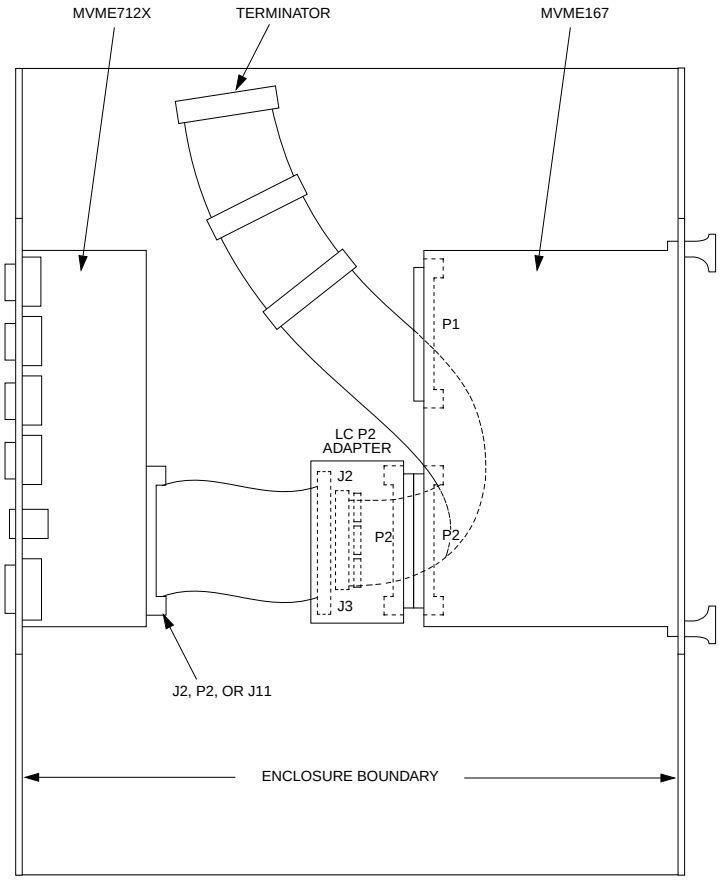


Figure 3-2. Typical Internal SCSI and Serial Port Connections

Equipment Connections

Some connection diagrams are in the *Single Board Computer Programmer's Reference Guide*.

The MVME712X transition modules and P2 adapter boards connect peripheral equipment to the MVME167 as shown in Table 3-10:

Table 3-10. Peripheral Connections

Equipment Type	Connect Through...
Console terminals, host computer systems, modems, or serial printers	EIA-232-D serial ports on the transition module
Parallel printers	Centronics printer port on the transition module
Optional internal modems (see the user's manual for your transition module for details)	Optional modem port, replacing serial port 2 on the transition module
Internal SCSI drives	Adapter board and transition module
External SCSI drives	SCSI interface connector on the transition module
Ethernet connections	Ethernet port on the transition module

Installing Transition Modules and Adapter Boards

Table 3-11. Transition Module and Adapter Board Installation Overview

Stage	What you will need to do...	Refer to...
1	Set jumpers on the transition module(s) and install SCSI terminators (if needed) on the P2 adapter board.	***Module Preparation*** in the user's manual for your transition module and adapter board, and *SCSI Bus Termination* on page 3-28
2	Connect and install the MVME712X transition module in the front or the rear of the chassis.	***Installation Instructions*** in the user's manual for your transition module and adapter board
3	Connect and install	
	❑ The P2 adapter board at the P2 connector on the backplane at the MVME167 slot.	***Installation Instructions*** in the user's manual for your transition module and adapter board
	❑ SCSI cable(s) from the P2 adapter board to the MVME712X transition module and internal SCSI devices.	***Module Preparation*** in the user's manual for your transition module and adapter board, and *SCSI Bus Termination* on page 3-28

Connecting Peripherals

The MVME167 mates with (optional) terminals or other peripherals at the EIA-232-D serial ports (marked SERIAL PORTS 1, 2, 3, and 4 on the MVME712X transition module), parallel port, SCSI ports, and LAN Ethernet port, as shown in Figure 3-4 on page 3-22.

Note Some cable(s) are not provided with the MVME712X module(s), and therefore are made or provided by the user. (Motorola recommends using shielded cables for all connections to peripherals to minimize radiation.)

Table 3-12. Peripheral Connection Procedures

Step	Action...	Refer to...
1	Connect and install any optional SCSI device cables from J3 on the P2 Adapter to internal devices and/or the MVME712B or MVME712M to external SCSI devices (typical configurations shown in Figure 3-2 on page 3-17 and Figure 3-3 on page 3-21).	The ***Single Board Computer Programmer's Reference Guide*** for some possible connection diagrams
2	Connect the terminal to be used as the 167Bug system console to the default debug EIA-232-D port at serial port 1 on the MVME712X transition module.	
3	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 MVME167 ports at powerup) After powerup, the baud rate of the debug port can be reconfigured by using the Port Format (**PF**) command of the 167Bug debugger.	
4	Connect devices such as a host computer system and/or a serial printer to the other EIA-232-D port connectors with the appropriate cables and configuration After powerup, these ports can be reconfigured by programming the MVME167 CD2401 Serial Controller Chip (SCC), or by using the 167Bug **PF** command.	
	Set up the device serial ports as described in Step 3. After powerup, the baud rate of the port can be reconfigured by using the Port Format (**PF**) command of the 167Bug debugger.	***Configuring a Port*** under the **PF** (Port Format) command in the ***Debugging Package for 68K CISC CPUs User's Manual***
5	Connect a parallel device, such as a printer, to the printer port on the MVME712X transition module. (You may also use a module such as the MVME335 for a parallel port connection.)	The ***Single Board Computer Programmer's Reference Guide*** for some possible connection diagrams

Note In order for high-baud rate serial communication between 167Bug 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, you should check the terminal to make sure XON/XOFF handshaking is enabled. Refer to ***Configuring a Port*** under the **PF** (Port Format) command in the ***Debugging Package for 68K CISC CPUs User's Manual***.

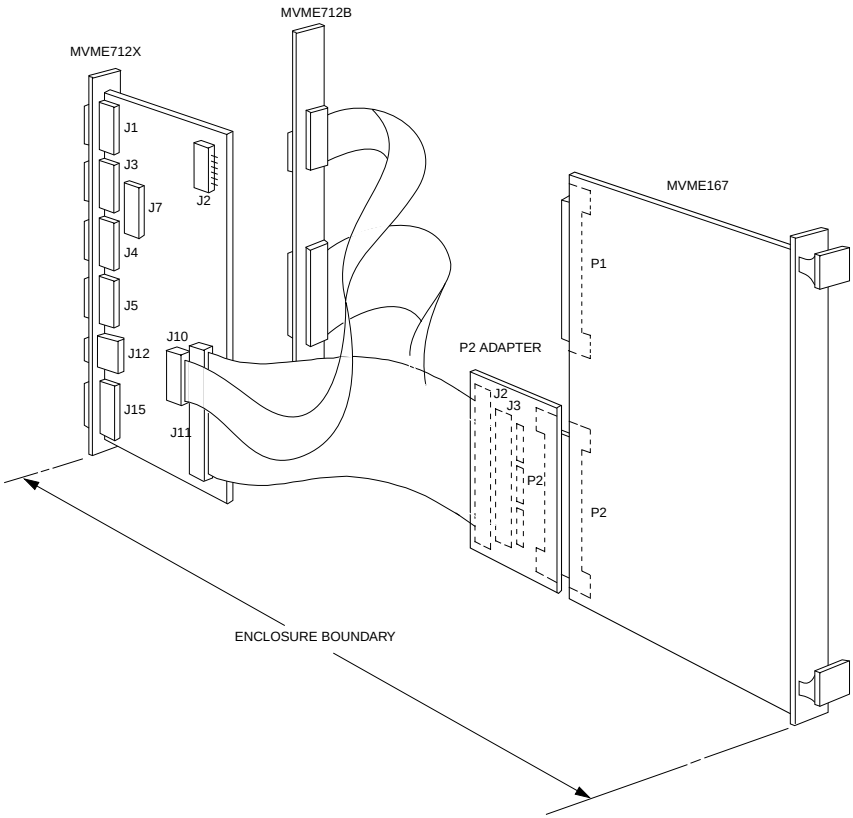


Figure 3-3. Using MVME712A/AM and MVME712B

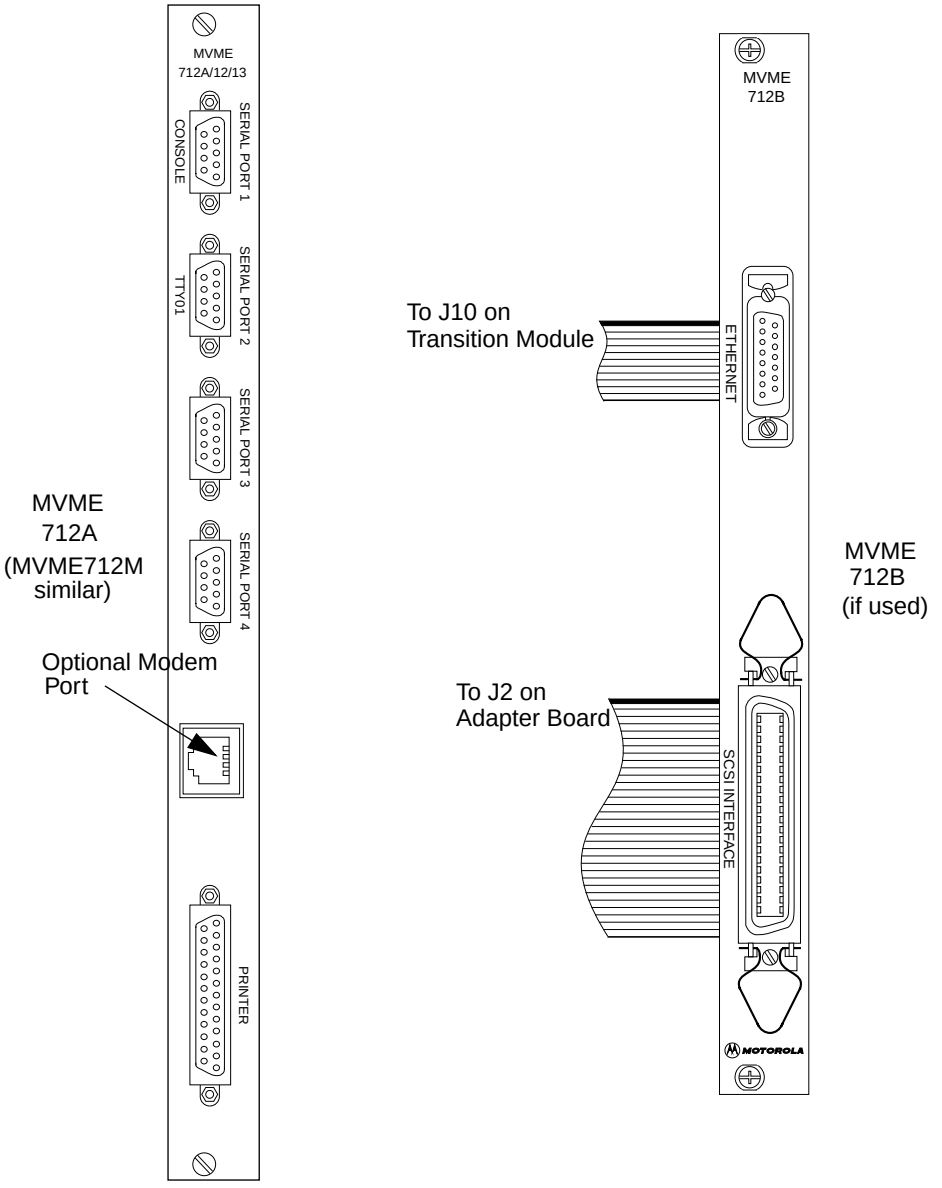


Figure 3-4. Typical Transition Module Peripheral Port Connectors

Completing the Installation

Table 3-13. Installation Completion Procedure

Step	Action...
1	Reassemble the chassis.
2	Reconnect the AC power.

Starting the System

After completing the preparation and installation procedures, you are ready to start up your system.

Table 3-14. System Startup Overview

Stage	What you will need to do...	Refer to...
1	Power up the system and note that the debugger prompt appears.	Page 3-24; *Starting Up 167Bug* on page 4-6; and the ***MVME167Bug Debugging Package User's Manual***.
2	Initialize the real-time clock.	Page 3-24.
3	Examine and/or change environmental parameters.	Page 3-24 and *Configure and Environment Commands* on page A-1.
4	Program the PCCchip2 and VMEchip2.	***System Considerations*** on page 3-26; *Memory Maps* on page 2-24; *ASICs* on page 2-8; and the *Single Board Computer Programmer's Reference Guide*.

Powering Up the System

The following table shows what takes place when you turn equipment power ON (depending on whether 167Bug is in Board Mode or System Mode):

If 167Bug is in...		
Board Mode	167Bug executes some self-checks and displays the debugger prompt `167-Bug>`	
System Mode	The system performs a selftest and tries to autoboot.	If the confidence test fails, the test aborts when the first fault occurs. If possible, an appropriate message is displayed, and control then returns to the menu (refer to Chapter 4, ***Debugger General Information***, and Chapter 5, ***Using the 167Bug Debugger*** for **ENV** and **MENU** commands).

Initializing the Real-Time Clock

The onboard real-time clock (RTC) is backed up with a self-contained battery. Before shipment of this board, the clock of the RTC device was stopped to preserve battery life.

The board's "Selftests" (ST) and operating systems require the clock of the RTC to be operating. Table 3-15 shows the steps required to initialize the RTC, depending on the mode.

Examining and/or Changing Environmental Parameters

Use the 167Bug's **ENV** command to verify the NVRAM (BBRAM) parameters, and optionally use **ENV** to make changes to the Environmental parameters. Refer to Appendix A for the Environment parameters.

Table 3-15. RTC Initialization Procedure

Step	Action	
	Board Mode	**System Mode**
1	Allow 167Bug to boot up normally.	Stop the auto-boot sequence by pressing the **<BREAK>** key. (If the system has already started and failed a confidence test in system mode, you should be in the debugger menu).
2	At the `167-Bug>` prompt, enter **TIME** to display the current date and time of day.	Select `(3)` from the debugger menu to get the debugger prompt.

3	At the `167-Bug>` prompt, use the **SET** command to initialize the RTC and to set the time and date. Use the following command line structure:
	SET `[<MMddyyhhmm>]|[<+/-CAL>;C]`
	For example:
	`167-Bug>SET 0522961037` <Return>
	`WED May 22 10:37:00.00 1996`
	`167-Bug>`
	Where the arguments are: MM=month, dd=day of the month, yy=year, hh=hour in "military" (24 hour) time, and mm=minutes.	

Programming the PCCchip2 and VMEchip2

See ***System Considerations*** below, and refer to *Memory Maps* on page 2-24, and the *Single Board Computer Programmer's Reference Guide* for details on your particular system environment.

System Considerations

Backplane Power Connections

The MVME167 needs to draw 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 MVME167 may not operate properly without its main board connected to P1 and P2 of the VMEbus backplane.

Memory Address Ranges

Whether the MVME167 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 certain address ranges. D8 and/or D16 devices in the system must be handled by the MC68040 software. Refer to the memory maps in the *Single Board Computer Programmer's Reference Guide* as listed in *Related Documentation* in Chapter 1.

DRAM Addressing

The MVME167 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 MVME167Bug firmware. This may be changed, by software, to any other base address. Refer to the *Single Board Computer Programmer's Reference Guide* for details.

Global Bus Timeout

If the MVME167 tries to access offboard resources in a non-existent location, and is not system controller, and if the system does not have a global bus timeout, the MVME167 waits forever for the VMEbus cycle to complete. This would cause the system to hang up.

Multiple Module Cage Configuration

Multiple MVME167s may be configured into a single VME card cage. In general, hardware multiprocessor features are supported.

GCSR Location Monitor Register

Other MPUs on the VMEbus can interrupt, disable, communicate with and determine the operational status of the CISC processor(s). One register of the GCSR set includes four bits which function as location monitors to allow one MVME167 processor to broadcast a signal to other MVME167 processors, if any. All eight of the GSCR registers are accessible from any local processor as well as from the VMEbus.

Ethernet LAN (+12 Vdc) Fuse

The MVME167 provides +12 Vdc power to the Ethernet LAN transceiver interface through a 1-amp fuse (F2) located on the MVME167. The +12V LED lights when +12 Vdc is available. The fuse is socketed and is located adjacent to diode CR1 near connector P1. If the Ethernet transceiver fails to operate, check the fuse. When using the MVME712M transition module, the yellow LED (DS1) on the MVME712M front panel lights when LAN power is available, indicating that the fuse is good.

Note Your MVME167 may have a poly-switch instead of a 1-amp fuse. If the Ethernet transceiver fails to operate and the poly-switch is open, power off the chassis and wait a while before re-applying power.

SCSI Bus Termination

- The MVME167 provides SCSI terminator power through a 1-amp fuse (F1) located on the P2 adapter board. The fuse is socketed. If the fuse is blown, 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 fuse should be checked.
- Because this board has no provision for SCSI termination, you must ensure that the SCSI bus is terminated properly. If you use a P2 adapter, the adapter has sockets (R1, R2, R3) for terminating the SCSI lines using three 8-pin SIP resistor networks.

Storage and the Real-Time Clock

For storage of this product, be sure the RTC is put into the power save mode. This will extend the life of the battery contained in this part. To put the part into the power save mode, use the **PS** command of the debugger. For example:

```
167-Bug>ps <Return>
(Clock is in Battery Save mode)
167-Bug>
```

Debugger General Information 4

This Chapter Covers

- An introduction to the MVME167Bug firmware package
- Booting and restarting 167Bug
- Disk input/output support capabilities
- Network support capabilities

Introduction to MVME167Bug

This section covers:

- Overview of M68000 firmware
- Description of 167Bug
- 167Bug implementation
- Memory requirements

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 BUG firmware currently used on all Motorola M68000-based CPU modules. The M68000 firmware family provides a high degree of functionality and user friendliness, and yet stresses portability and ease of maintenance. This member of the M68000 firmware family is implemented on the MVME167 CISC Single Board Computer, and is known as the MVME167Bug, or just 167Bug.

Description of 167Bug

The 167Bug package, MVME167Bug, is a powerful evaluation and debugging tool for systems built around the MVME167 CISC-based microcomputers.

167Bug consists of three parts:

- The “debugger” or “167Bug“; a command-driven user-interactive software debugger, described in Chapter 5
- A command-driven diagnostic package for the MVME167 hardware
- A user interface which accepts commands from the system console terminal

Command Facilities

Facilities are available for loading and executing user programs under complete operator control for system evaluation.

167Bug includes commands for these tasks:

- Display and modification of memory
- Breakpoint and tracing capabilities
- A powerful assembler/disassembler useful for patching programs
- A self-test at powerup feature which verifies the integrity of the system.

Trap #15 System Calls

Various 167Bug routines that handle I/O, data conversion, and string functions are available to user programs through the TRAP #15 system calls.

Debugger or Diagnostic Directories

When using 167Bug, you operate out of either the debugger directory or the diagnostic directory.

If you are in ...	With the prompt ...	You have available ...
The debugger directory	`167-Bug>`	All of the debugger commands
The diagnostic directory	`167-Diag>`	All of the diagnostic commands as well as all of the debugger commands

You may switch between directories by using the Switch Directories (**SD**) command.

You may examine the commands in the current directory by using the Help (**HE**) command.

Keyboard Control

167Bug is command-driven; it performs its various operations in response to user commands entered at the keyboard. When you enter a command, 167Bug executes the command and the `167-Bug>` 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 167Bug, depending on the outcome of the user program.

Similarity to other Motorola Debugging Firmware

If you have used one or more of Motorola's other debugging packages, you will find the CISC 167Bug very similar. Some 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.

167Bug Implementation

MVME167Bug 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, 167Bug is contained in two of the four 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 MC68040 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.

Memory Requirements

The program portion of 167Bug is approximately 512KB of code, consisting of download, debugger, and diagnostic packages and contained entirely in EPROM. The EPROM sockets on the MVME167 are mapped starting at location $FF800000.

167Bug 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 167Bug stack and static variable space and the rest is reserved as user space.

Whenever the MVME167 is reset, the target PC is initialized to the address corresponding to the beginning of the user space, and the target stack pointers are initialized to addresses within the user space, with the 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 167Bug initial stack.

Booting and Restarting 167Bug

This section covers the following tasks:

- Starting up 167Bug
- Autoboot
- ROMboot
- Network boot
- Restarting the system

Starting Up 167Bug

1. Verify that the MVME167 is properly installed and operating as described in Table 3-1 on page 3-2.
2. Power up the system. 167Bug executes some self-checks and displays the debugger prompt `167-Bug>` (if 167Bug is in Board Mode). However, if the **ENV** command (Appendix A) has put 167Bug in System Mode, the system performs a selftest and tries to autoboot. Refer to the **ENV** and **MENU** commands listed in Table 5 -1.

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

Autoboot

Autoboot is a software routine that is contained in the 167Bug EPROMs to provide an independent mechanism for booting an operating system.

Autoboot Sequence

1. The 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.
2. If a valid bootable device is found, a boot from that device is started.

   The controller scanning sequence goes from the lowest controller Logical Unit Number (LUN) detected to the highest LUN detected. (Controllers, devices, and their LUNs are listed in Appendix B).
3. At powerup, Autoboot is enabled, and if the drive and controller numbers encountered are valid, the system console displays the following message:

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

4. Following this message there is a delay to allow you an opportunity to abort the Autoboot process if you wish. To gain control without Autoboot, you can press the BREAK key or the software ABORT or RESET switches.

5. Then the actual I/O begins: the program pointed to within the volume ID of the media specified is loaded into RAM and control passed to it.

Autoboot is controlled by parameters contained in the **ENV** command. These parameters allow the selection of specific boot devices and files, and allow programming of the Boot delay. Refer to the **ENV** command in Appendix A for more details.

ROMboot

There are two spare EPROM sockets, XU3 and XU4, available to carry user-programmed EPROMs. Therefore, you do not have to reprogram the 167Bug EPROMs in order to implement the ROMboot feature.

One use of ROMboot might be resetting SYSFAIL* on an unintelligent controller module. The **NORB** command disables the function.

ROMboot Sequence

1. ROMboot is configured/enabled and executed at powerup (optionally also at reset) in one of two ways:

   a. By the Environment (**ENV**) command (refer to Appendix A).

   b. By the **RB** command assuming there is valid code in the EPROMs (or optionally elsewhere on the module or VMEbus) to support it.

2. If ROMboot code is installed, a user-written routine is given control (if the routine meets the format requirements).

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

Requirement ...	Optionally, with the ENV command ...
Power must have just been applied.	Change this to respond to any reset.
Your routine must be located within the MVME167 ROM memory map.	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 powerup.	

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 167Bug EPROMs that provides a mechanism for booting an operating system using a network (local Ethernet interface) as the boot device.

Network Boot Sequence

1. 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.
2. If a valid bootable device is found, a boot from that device is started.

   The controller scanning sequence goes from the lowest controller Logical Unit Number (LUN) detected to the highest LUN detected. (Refer to Appendix C for default LUNs.)

3. At powerup, Network Boot is enabled, and providing the drive and controller numbers encountered are valid, the following message is displayed 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 Network Boot process if you wish. To gain control without Network Boot, you can press the BREAK key or the software ABORT or RESET switches.

4. Then the actual I/O begins: the program pointed to within the volume ID of the media specified is loaded into RAM and control passed to it.

Network Auto Boot is controlled by parameters contained in the **NIOT** and **ENV** commands. These parameters allow the selection of specific boot devices, systems, and files, and allow programming of the Boot delay. Refer to the **ENV** command in Appendix A.

Restarting the System

You can initialize the system to a known state in three different ways: reset, abort, and break. Each has characteristics which make it more appropriate than the others in certain situations.

The debugger has a special feature which can be used in the event your setup/operation parameters are corrupted or do not meet a sanity check. This feature:

- Is activated by pressing the RESET and ABORT switches at the same time, then releasing the RESET switch before the ABORT switch.
- Instructs 167Bug to use the default setup/operation parameters in ROM instead of your setup/operation parameters in NVRAM.

Refer to the **ENV** command (Appendix A) for the ROM defaults.

Reset

Pressing and releasing the MVME167 front panel RESET switch initiates a system reset.

Reset must be used if the processor ever halts, or if the 167Bug environment is ever lost (vector table is destroyed, stack corrupted, etc.).

- COLD and WARM reset modes are available.
- By default, 167Bug is in COLD mode.

During COLD reset:

1. A total system initialization takes place, as if the MVME167 had just been powered up.
2. All static variables (including disk device and controller parameters) are restored to their default states.
3. The breakpoint table and offset registers are cleared.
4. The target registers are invalidated.
5. Input and output character queues are cleared.
6. Onboard devices (timer, serial ports, etc.) are reset, and
7. The *first* two serial ports are reconfigured to their default state.

During WARM reset:

1. The 167Bug variables and tables are preserved, as well as the target state registers and breakpoints.

Abort

Abort is invoked by pressing and releasing the ABORT switch on the MVME167 front panel.

Abort should be used to regain control if the program gets caught in a loop, etc.

Whenever abort is invoked when 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, register contents, etc., help to pinpoint the malfunction.

Abort Sequence

Pressing and releasing the ABORT switch does the following:

1. Generates a local board condition which may interrupt the processor if enabled.
2. Displays the target registers on the screen, reflecting the machine state at the time the ABORT switch was pressed.
3. Removes any breakpoints installed in the user code and keeps the breakpoint table intact.
4. Returns control to the debugger.

Break

A “Break” is generated by pressing and releasing the BREAK key on the console 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.

Many times it may be 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.

Break Sequence

1. Removes any breakpoints in your code and keeps the breakpoint table intact.
2. Takes a snapshot of the machine state if the function was entered using SYSCALL. This machine state is then accessible to you for diagnostic purposes.

SYSFAIL* Assertion/Negation

Upon a reset/powerup 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. Refer to Appendix A.

MPU Clock Speed Calculation

The clock speed of the microprocessor is calculated and checked against a user definable parameter contained in NVRAM (refer to the **CNFG** command in Appendix A). If the check fails, a warning message is displayed.

Disk I/O Support

167Bug can initiate disk input/output by communicating with intelligent disk controller modules over the VMEbus.

This section covers:

- Blocks Versus Sectors
- Device Probe Function
- Disk I/O via 167Bug Commands
- Disk I/O via 167Bug System Calls
- Default 167Bug Controller and Device Parameters
- Disk I/O Error Codes

Disk Support Facilities

Disk support facilities built into 167Bug 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.

Parameter Tables

Parameters such as the address where the module is mapped and the type and number of devices attached to the controller module are kept in tables by 167Bug. Default values for these parameters are assigned at powerup and cold-start reset, but may be altered as described in the section on default parameters, later in this chapter.

Supported Controllers

Appendix B contains a list of the controllers presently supported, as well as a list of the default configurations for each controller.

Blocks Versus Sectors

The logical block defines the unit of information for disk devices. A disk is viewed by 167Bug 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. The block size can be changed 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, the start and size of the transfer is specified in blocks. 167Bug translates this into an equivalent sector specification, which is then passed 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 done whenever a specified device is accessed; i.e., when system calls .DSKRD, .DSKWR, .DSKCFIG, .DSKFMT, and .DSKCTRL, and debugger commands **BH**, **BO**, **IOC**, **IOP**, **IOT**, **MAR**, and **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 167Bug Commands

These following 167Bug commands 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 167Bug so that the next disk command defaults to use the same controller and device.

IOI (Input/Output Inquiry)

This command is used to probe the system for all possible CLUN/DLUN combinations and display inquiry data for devices which support it. The device descriptor table only has space for 16 device descriptors; with the **IOI** command, you can view the table and clear it if necessary.

IOP (Physical I/O to Disk)

IOP allows you to read or write blocks of data, or to format the specified device in a certain way. **IOP** creates a command packet from the arguments you have specified, and 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 see 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 look at 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, and then transfers control to it.

BH (Bootstrap and Halt)

BH reads an operating system or control program from a specified device into memory, and then returns control to 167Bug. It is used as a debugging tool.

4

Disk I/O via 167Bug System Calls

All operations that actually access the disk are done directly or indirectly by 167Bug 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 are provided to allow user programs to do disk I/O:

.DSKRD	Disk read. System call to read blocks from a disk into memory.
.DSKWR	Disk write. System call to write blocks from memory onto a disk.
.DSKCFIG	Disk configure. This function allows you to change the configuration of the specified device.
.DSKFMT	Disk format. This function allows you to send a format command to the specified device.
.DSKCTRL	Disk control. This function is used 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.

Controller Command Packets

To perform a disk operation, 167Bug 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 do disk I/O accept a generalized (controller-independent) packet format as an argument, and translate it into a controller-specific packet, which is then sent 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 be given 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 it is sent to. Refer to documentation on the particular controller module for the format of its packets, and for using the **IOC** command.

Default 167Bug Controller and Device Parameters

167Bug initializes the parameter tables for a default configuration of controllers and devices (refer to Appendix B). 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 the parameter tables:

Using ...	When you invoke one of these commands ...	Change status is ...	If a cold-start reset occurs ...
Command BO or BH	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.	Temporary	The default parameter information is written back into the tables.
Command IOT	You can use this command to reconfigure the parameter table manually for any controller and/or device that is different from the default.	Temporary	The default parameter information is written back into the tables.
The source code	In the source code, you may change the permanent configuration files and rebuild 167Bug so that it has different defaults.	Permanent until changed again.	

Disk I/O Error Codes

167Bug 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 is executed in two distinct phases.

- The first phase allows the diskless remote node to discover its network identify and the name of the file to be booted.
- The second phase has the diskless remote node reading the boot file across the network into its memory.

The various modules 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 the reception of packets, the transmission of packets, receive buffer flushing, and interface initialization.

This module ensures that the packaging and unpackaging of Ethernet packets is done 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, and waiting for an answer. The RARP server fills an Ethernet reply packet up with the target's Internet Address and sends it.

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.

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, and 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 and to sequence the booting process. The booting sequence consists of two phases: the first phase is labeled "address determination and bootfile selection" and the second phase is labeled "file transfer". The first phase will utilize the RARP/BOOTP capability and the second phase will utilize the TFTP capability.

Network I/O Error Codes

167Bug returns an error code if an attempted network operation is unsuccessful.

Multiprocessor Support

The MVME167 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:

- The Multiprocessor Control Register (MPCR) Method
- The Global Control and Status Register (GCSR) Method

Either method can be enabled/disabled by the **ENV** command as its Remote Start Switch Method (refer to Appendix A).

Multiprocessor Control Register (MPCR) Method

A remote processor can initiate program execution in the local MVME167 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:

Base Address + $800	*	N/A	N/A	N/A	(MPCR)

MPCR Status Codes

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

- Status returned (from the 167Bug)
- Command set by the bus master (job requested by some processor)

The status codes that may be returned from 167Bug 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 is executing.

The command code that may be set by the bus master is:

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

Multiprocessor Address Register (MPAR)

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 physical address (as viewed from the local processor) at which execution for this processor is to begin if the MPCR contains a G or B. The MPAR is organized as follows:

Base Address + $804	*	*	*	*	(MPAR)

MPCR Powerup sequence

1. At powerup, the debug monitor self-test routines initialize RAM, including the memory locations used for multi-processor support ($800 through $807).
2. The MPCR contains $00 at powerup, indicating that initialization is not yet complete.

3. 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 see 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.)

4. 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 every 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.

Global Control and Status Register (GCSR) Method

A remote processor can initiate program execution in the local MVME167 dual-port RAM by issuing a remote **GO** command using the VMEchip2 Global Control and Status Registers (GCSR).

1. The remote processor places the MVME167 execution address in general purpose registers 0 and 1 (GPCSR0 and GPCSR1).
2. The remote processor then sets bit 8 (SIG0) of the VMEchip2 LM/SIG register.
3. This causes the MVME167 to install breakpoints and begin 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 MVME167. 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

Included in the 167Bug package is a complete set of hardware diagnostics intended for testing and troubleshooting of the MVME167. These diagnostics are completely described in the *MVME167Bug Debugging Package User's Manual*.

- In order to use the diagnostics, you must switch directories to the diagnostic directory.
- If you are in the debugger directory, you can switch to the diagnostic directory by entering the debugger command Switch Directories (**SD**). The diagnostic prompt

  ```
  167-Diag>
  ```

  should appear.

Table 4-1. Diagnostic Monitor Commands/Prefixes

Command/ Prefix	Description
AEM	Append Error Messages Mode
CEM	Clear Error Messages
CF	Test Group Configuration Parameters Editor
DE	Display Error Counters
DEM	Display Errors
DP	Display Pass Count
HE	Help
HEX	Help Extended
LA	Loop Always Mode
LC	Loop Continuous Mode
LE	Loop on Error Mode
LF	Line Feed Suppression Mode
LN	Loop Non-Verbose Mode
MASK	Display/Revise Self Test Mask

Table 4-1. Diagnostic Monitor Commands/Prefixes

Command/ Prefix	Description
NV	Non-Verbose Mode
SD	Switch Directories
SE	Stop on Error Mode
ST	Selftest
ZE	Clear (Zero) Error Counters
ZP	Zero Pass Count

Table 4-2. Diagnostic Utilities

Command	Description
WL	Write loop enable
RL	Read loop enable

167Bug Diagnostic Test Groups

Refer to the *MVME167Bug Debugging Package User's Manual* for complete descriptions of the diagnostic routines available and instructions on how to invoke them. Note that 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.

Table 4-3. Diagnostic Test Groups

Test Set	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
VME2	VME Interface ASIC VMEchip2 Tests
LANC	LAN Coprocessor (Intel 82596) Tests
NCR	NCR 53C710 SCSI I/O Processor Tests
DCAC	MC68040 Data Cache Tests
MMU	MC68040 MMU Tests

Using the 167Bug Debugger 5

This Chapter Covers

- Entering debugger command lines
- Entering and debugging programs
- Calling system utilities from user programs
- Preserving the debugger operating environment
- Floating point support
- The 167Bug debugger command set

Entering Debugger Command Lines

167Bug is command-driven and performs its various operations in response to user commands entered at the keyboard. When the debugger prompt

```
167-Bug>
```

appears on the terminal screen, then the debugger is ready to accept commands.

Terminal Input/Output Control

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 below.

Note The presence of the upward 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.
^H	(backspace)	The cursor is moved back one position.
Delete key	(delete)	Performs the same function as **^H**.
^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.

When observing output from any 167Bug 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 167Bug, 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.

When a command is entered, 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".

Debugger Command Syntax

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

- The command identifier (i.e., **MD** or **md** for the Memory Display command). Note that either upper- or lowercase is allowed.
- A port number if the command is set up to work with more than one port.
- At least one intervening space before the first argument.
- Any required arguments, as specified by the command.
- 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 (<<), or shift right (>>).

Numeric values may be expressed in either hexadecimal, decimal, octal, or binary by immediately preceding them with the proper base identifier.

Base	Identifier	Examples
Hexadecimal	$	$FFFFFFFF
Decimal	&	&1974, &10-&4
Octal	@	@456
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 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 167Bug is similar to the one accepted by the MC68040 one-line assembler. All control addressing modes are allowed. An "address + offset register" mode is also provided.

Address Formats

Table 5-1 summarizes the address formats that are acceptable for address parameters in debugger command lines.

Table 5-1. 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, pre-decrement)
(*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 and 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          END
     ******  TOTAL ERRORS     0——
     ******  TOTAL WARNINGS   0——
```

The above program was loaded at address $0001327C.

The disassembled code is shown next:

```
167Bug>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
167Bug>
```

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

```
167Bug>OF R0
R0 =00000000 00000000? 1327C. <CR>
167Bug>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
167Bug>
```

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

Port Numbers

Some 167Bug commands give you the option to choose the port to be used to input or output. Valid port numbers which may be used for these commands are as follows:

1. MVME167 EIA-232-D Debug (Terminal Port 0 or 00) (PORT 1 on the MVME167 P2 connector). Sometimes known as the "console port", it is used for interactive user input/output by default.
2. MVME167 EIA-232-D (Terminal Port 1 or 01) (PORT 2 on the MVME167 P2 connector). Sometimes known as the "host port", this is the default for downloading, uploading, concurrent mode, and transparent modes.

Note These logical port numbers (0 and 1) are shown in the pinouts of the MVME167 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:

- Create the program with the assembler/disassembler
- Download an S-record object file
- Read the program from disk

Creating a Program with the Assembler/Disassembler

You can create a program using the Memory Modify (**MM**) command with the assembler/disassembler option.

1. Enter the program one source line at a time.
2. After each source line is entered, it is assembled and the object code is loaded to memory.

Refer to the *Debugging Package for Motorola 68K CISC CPUs User's Manual* for complete details of the 167Bug Assembler/Disassembler.

Downloading an S-Record Object File

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 167Bug **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 MVME167 port 1. (Hardware configuration details are provided in *Connecting Peripherals* on page 3-19.) The file is downloaded from the host to MVME167 memory by the Load (**LO**) command.

Read the Program from Disk

Another way to enter a program is by reading the program from disk, using one of the disk commands (**BO**, **BH**, **IOP**). Once the object code has been loaded into memory, you can set breakpoints if desired and run the code or trace through it.

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 167Bug routines via one of the MC68040 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. Topics covered include:

- 167Bug Vector Table and workspace
- Hardware functions
- Exception vectors used by 167Bug

167Bug uses certain of the MVME167 onboard resources and may also use offboard system memory to contain temporary variables, exception vectors, etc. If you disturb resources upon which 167Bug 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.

167Bug Vector Table and Workspace

As described in *Memory Requirements* on page 4-5, 167Bug needs 64KB of read/write memory to operate.

167Bug reserves ...	For ...
1024-byte area	A user program vector table area
1024-byte area	An exception vector table for the debugger itself to use
Space for static variables and initializes these static variables to predefined default values.	
Space for the system stack then 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.

Refer to *Memory Requirements* on page 4-5 to determine how to dictate the location of the reserved memory areas.

Examples

- 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 and a host. 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 167Bug

The exception vectors used by the debugger are listed below. 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.

Table 5-2. Exception Vectors Used by 167Bug

Vector Offset	Exception	167Bug 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

Table 5-2. Exception Vectors Used by 167Bug (Continued)

Vector Offset	Exception	167Bug Facility
$BC	TRAP #15	System calls
NOTE	Level 7 interrupt	ABORT pushbutton
NOTE	Level 7 interrupt	AC Fail
$DC	FP Unimplemented Data Type	Software emulation and data type conversion of floating point data.

Note These depend on what the Vector Base Register (VBR) is set to in the VMEchip2.

When the debugger handles one of the exceptions listed in Table 5-2, 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.

Example: Trace one instruction using debugger.

```
167Bug>RD
PC   =00010000 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =0000DFFC MSP  =0000EFFC ISP* =0000FFFC SFC  =0=F0
DFC  =0=F0     CACR =0=........
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   =00000000 A6   =00000000 A7   =0000FFFC
00010000 203C0000 0001    MOVE.L     #$1,D0
167Bug>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
167Bug>
```

Exception Vector Tables

Notice in the preceding example 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 167Bug or it may create a separate exception vector table of its own. The two following sections detail these two methods.

Using 167Bug Target Vector Table

The 167Bug 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 trace commands (**T**, **TC**, or **TT**)
- Automatically with the **BO** command.

The start address of this target vector table area is the base address ($00) of the debugger memory. This address is loaded into the target-state VBR at powerup and cold-start reset and can be observed by using the **RD** command to display the target-state registers immediately after powerup.

The 167Bug initializes the target vector table with the debugger vectors listed in Table 5-2 on page 5-13 and fills the other vector locations with the address of a generalized exception handler (refer to the *167Bug Generalized Exception Handler* on page 5-17 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 5-2 are overwritten then the accompanying debugger functions are lost.

The 167Bug 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 at the new vector table. In order to use the debugger facilities you can copy the proper vectors from the 167Bug vector table into the corresponding vector locations in your program vector table.

The vector for the 167Bug generalized exception handler (described in detail in *167Bug Generalized Exception Handler* on page 5-17 in this chapter) may be copied from offset $3C (uninitialized interrupt) 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.

```
*
***  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.
```


167Bug Generalized Exception Handler

The 167Bug has a generalized exception handler which it uses to handle all of the exceptions not listed in Table 5-2 on page 5-13. This guarantees that all exceptions have an interrupt handler assigned to handle it. The handler stops target code execution and displays the register contents.

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     $3C(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     $8(A0),$8(A1)      Copy bus error vector.
        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 turn out 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 167Bug target program vector table (which your program saved), yielding the address of the 167Bug exception vector. The program then jumps to the address stored at this vector location, which is the address of the 167Bug 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.

Floating Point Support

The floating point unit (FPU) of the MC68040 microprocessor chip is supported in 167Bug. For MVME167Bug, the commands **MD**, **MM**, **RM**, and **RS** have been extended to allow display and modification of floating point data in registers and in memory. Floating point instructions can be assembled and 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, double, extended precision, or 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 takes 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 takes 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 takes 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 takes 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 and 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 MC68040 floating point unit, refer to the *MC68040 Microprocessor User’s Manual*.

The 167Bug Debugger Command Set

The 167Bug debugger commands are summarized in Table 5-3. The command syntax is shown using the symbols explained earlier in this chapter. The **CNFG** and **ENV** commands are explained in Appendix A. Controllers, devices, and their LUNs are listed in Appendix B or Appendix C. All other command details are explained in the *MVME167Bug Debugging Package User's Manual*.

Table 5-3. 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* [*del 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**]

Table 5-3. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
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*'}
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** [*port*] [*addr*] [**; [X] [C] [T]**] [*=text*]
MA	Macro Define/Display	**MA** [*name* | **; L**]
NOMA	Macro Delete	**NOMA** [*name*]
MAE	Macro Edit	**MAE** *name del line#* [*del string*]
MAL	Enable Macro Expansion Listing	**MAL**
NOMAL	Disable Macro Expansion Listing	**NOMAL**
MAW	Save Macros	**MAW** [*controller LUN*] [*del* [*device LUN*] [*del block #*]]

Table 5-3. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
MAR	Load Macros	**MAR** [*controller LUN*] [*del* [*device LUN*] [*del block #*]]
MD	Memory Display	**MD** [**S**] *addr* [**:***count* | *del addr*] [**; [B | W | L | S | D | X | P | DI]**]
MENU	Menu	**MENU**
MM	Memory Modify	**MM** *addr* [**; [[B | W | L | S | D] [A] [N]] | [DI]**]
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 del device-LUN del source-IP del destination-IP* [*del n-packets*]
OF	Offset Registers Display/Modify	**OF** [**R***n* [**; A**]]
PA	Printer Attach	**PA** [*port*]
NOPA	Printer Detach	**NOPA** [*port*]
PF	Port Format	**PF** [*port*]
NOPF	Port Detach	**NOPF** [*port*]
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**]
REMOTE	Connect the Remote Modem to CSO	**REMOTE**

Table 5-3. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
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* | *n*; **C**
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** [*port*] [*del* ESCAPE]
TT	Trace to Temporary Breakpoint	**TT** *addr*
VE	Verify S-Records Against Memory	**VE** [*port*] [*addr*] [; [**X**][**C**]] [=*text*]
VER	Display Revision/Version	**VER** [; **E**]
WL	Write Loop	**WL** *addr del data* [; **B**|**W**|**L**]

Configure and Environment Commands

A

This Appendix Covers

- Configuring the board information block
- Setting the environment to Bug/Operating System
- Environment command parameters

Configure Board Information Block

CNFG [;[I][M]]

This command is used to display and configure the board information block. This block is resident within the Non-Volatile RAM (NVRAM). Refer to the *MVME167 Single Board Computer User's Manual* for the actual location.

The information block contains various elements detailing specific operation parameters of the hardware. The *MVME167 Single Board Computer User's Manual* describes the elements within the board information block, and lists the size and 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.

```
167-Bug>cnfg
Board (PWA) Serial Number = "000000061050"
Board Identifier          = "MVME167-03      "
Artwork (PWA) Identifier  = "01-W3826B03A    "
MPU Clock Speed           = "2500"
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       = "        "
167-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. A 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. These parameters are initialized by the factory, and correct board operation relies upon these parameters.

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

Setting Environment to Bug/Operating System

ENV [;[**D**]]

The Environment **(ENV**) command allows you to interactively view and 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 modified parameters are loaded into NVRAM along with checksum data. The operational parameters that have been modified 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 parameters to be configured are listed in the following table:

Table A-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 on a 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 00.
Auto Boot Device LUN	00	LUN of a disk/tape device currently supported by the Bug. Default is 00.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
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.
Auto Boot Default String [NULL for an empty 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	0	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.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Network Auto Boot Controller LUN	00	LUN of a network controller module currently supported by the Bug. Default is 00.
Network Auto Boot Device LUN	00	LUN of a network device currently supported by the Bug. Default is 00.
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 A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Network Auto Boot Configuration Parameters Pointer (NVRAM)	00000000	This is the address where the network interface configuration parameters are to be saved 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 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.
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-167 environment, each MVME167 could be set to start its work page at a unique address to allow multiple debaters to operate simultaneously from the same memory.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
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 MVME167. 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.
Memory Search Delay Address	FFFFCE0F	The process of using the Memory Search Delay Address was implemented on the MVME188. It has not been used on the MVME167.
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.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
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 MVME167 memory mezzanine board. Default is the calculated size of the memory board.
Slave address decoders setup. The slave address decoders are used to allow another VMEbus master to access a local resource of the MVME167. There are two slave address decoders. They are set up as follows.		
Slave Enable #1 [Y/N]	Y	Yes, Setup and enable the Slave Address Decoder #1.
Slave Starting Address #1	00000000	Base address of the local resource that is accessible by the VMEbus, as viewed 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, as viewed 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 the local resource that is associated with the starting and ending address selection from the previous questions. Default is $0.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Slave Address Translation Select #1	00000000	This register defines which bits of the Translation Address are significant. A logical one "1" indicates significant address bits, logical zero "0" is non-significant. The non-significant bits will come from the VMEbus address that accesses the local resource. Normally, MS bits will be set, down to the size of memory to be accessed. Default is $0.
Slave Control #1	03FF	Defines the access restriction for the address space defined with this slave address decoder. Default is $03FF. See description of VMEchip2 bits in *Single Board Computer Programmer's Reference Guide*.
Slave Enable #2 [Y/N]	Y	Yes, Setup and enable the Slave Address Decoder #2.
Slave Starting Address #2	FFE00000	Base address of the local resource that is accessible by the VMEbus, as viewed by the VMEbus. Default is the base address of static RAM, $FFE00000.
Slave Ending Address #2	FFE1FFFF	Ending address of the local resource that is accessible by the VMEbus, as viewed 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	Works the same as Slave Control #1. Default is $01EF.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Master Enable #1 [Y/N]	Y	Yes, Setup 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	Works the same as Slave Control #1. 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 $0.
Master Ending Address #2	00000000	Ending address of the VMEbus resource that is accessible from the local bus. Default is $0.
Master Control #2	00	Works the same as Slave Control #1. Default is $00.
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	Works the same as Slave Control #1. Default is $00.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
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. This will be the local bus address. Default is $0.
Master Ending Address #4	00000000	Ending address of the VMEbus resource that is accessible from the local bus. This will be the local bus address. 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 the 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 Translation Address are significant. A logical one "1" indicates significant address bits, logical zero "0" is non-significant. The non-significant bits will come from the local bus address that accesses the VMEbus. Normally, MS bits will be set, down to the size of memory to be accessed. Default is $0.
Master Control #4	00	Works the same as Slave Control #1. Default is $00.
Short I/O (VMEbus A16) Enable [Y/N]	Y	Yes, Enable the Short I/O Address Decoder.
Short I/O (VMEbus A16) Control	01	Works the same as Slave Control #1. Default is $01.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
F-Page (VMEbus A24) Enable [Y/N]	Y	Yes, Enable the F-Page Address Decoder.
F-Page (VMEbus A24) Control	02	Works the same as Slave Control #1. Default is $02.
ROM Speed Bank A Code ROM Speed Bank B Code	05 05	Used to set up the ROM speed. Default is $05 = 165 ns (25 MHz MVME187) or $04=145 ns (33 MHz MVME187).
Static RAM Speed Code	01	Used to set up the SRAM speed. Default is $01 = 125 ns (25 MHz MVME167) or $00=115 ns (33 MHz MVME187)
PCC2 Vector Base VMEC2 Vector Base #1 VMEC2 Vector Base #2	05 06 07	Base interrupt vector for the component specified. Default: PCCchip2 = $05, VMEchip2 Vector 1 = $06, VMEchip2 Vector 2 = $07.
VMEC2 GCSR Group Base Address	CC	Specifies the group address ($FFFFXX00) in Short I/O for this board. Default = $CC.
VMEC2 GCSR Board Base Address	00	Specifies the base address ($FFFFCCXX) in Short I/O for this board. Default = $00.
VMEbus Global Time Out Code	01	This controls the VMEbus timeout when this MVME167 is the system 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.

Disk/Tape Controller Data

B

Disk/Tape Controller Modules Supported

The following VMEbus disk/tape controller modules are supported by the 167Bug. The default address for each controller type is First Address and the controller can be addressed by First CLUN during commands **BH**, **BO**, or **IOP**, or during 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 (MVME167)	$00	--	--	--
MVME320 - Winchester/Floppy Controller	$11	$FFFFB000	$12	$FFFFAC00
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

B

Disk/Tape Controller Default Configurations

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

CISC Single Board Computers -- 7 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 -- 4 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 -- 4 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 -- 9 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 -- 14 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 -- 1 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, and 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

IOT Parameter	Floppy Types and Formats						
	DSDD5	**PCXT8**	**PCXT9**	**PCXT9_3**	**PCAT**	**PS2**	**SHD**
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
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/HD	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.

Network Controller Modules Supported

The following VMEbus network controller modules supported by MVME167Bug are shown in Table C-1. The default address for each type and position is shown to indicate where the controller must reside to be supported by MVME167Bug.

The CLUNs and DLUNs are used in conjunction with the following debugger commands and debugger system calls:

Debugger Commands	Debugger System Calls
NBH	.NETRD
NBO	.NETWR
NIOC	.NETFOPN
NIOP	.NETFRD
NIOT	.NETCFIG
NPING	.NETCTRL
NAB	

The controllers are accessed via the CLUNs and DLUNs specified in the following table:

Table C-1. Network Controller Access Data

Controller Type	CLUN	DLUN	Address	Interface Type
MVME167	$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

Table C-1. Network Controller Access Data (Continued)

Controller Type	CLUN	DLUN	Address	Interface Type
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 MVME167: Solving Startup Problems

D

- 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 D-1. 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, in Chapter 3. 5. Review the Installation and Startup procedures, in Chapter 3. The step-by-step powerup routine for your board is on page 3-17. Try it.
	B. If the LEDs are lit, the board may be in the wrong slot.	1. The CPU board should be in the first (leftmost) slot if it is to be the system controller. 2. The "system controller" function requires that header J2 be set properly. See Chapter 3.
	C. The system console terminal may be configured wrong.	Configure the system console terminal according to the instructions in Chapter 3.

Table D-1. Troubleshooting Steps (Continued)

Condition	Possible Problem	Try This:
II. There is a display on the terminal, but input from the keyboard has no effect.	A. The keyboard may be connected incorrectly.	Recheck the keyboard 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 **^S** Also, a HOLD LED may be lit on the keyboard.	Press the HOLD or PAUSE key again. If this does not free up the keyboard, type in **^Q** (Hold down the CONTROL key and type a "Q")

Table D-1. Troubleshooting Steps (Continued)

Condition	Possible Problem	Try This:
III. Debug prompt 167-Bug> does not appear at powerup, and the board does not auto boot.	A. Debugger EPROM may be missing.	1. Disconnect *all* power from your system. 2. Check that the proper debugger EPROM is installed per this manual. 3. Reconnect power. **Caution** — Performing the next step will change some parameters that may affect your system operation. 4. Restart the system by "double-button reset": press the RESET and ABORT switches at the same time; release RESET first, wait five 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 D-1. Troubleshooting Steps (Continued)

Condition	Possible Problem	Try This:
IV. Debug prompt 167-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 in **set mmddyyhhmm <Return>** 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 <Return>** This sets up the default parameters for the debugger environment. 3. When prompted to Update Non-Volatile RAM, type in **y <Return>** 4. When prompted to Reset Local System, type in **y <Return>** 5. After clock speed is displayed, immediately (within five seconds) press the RETURN key **<Return>** or BREAK to exit to System Menu. Then enter a 3 "Go to System Debugger" and press the RETURN key **3 <Return>** Now the prompt should be 167-Diag>
	B. There may be some fault in the board hardware.	

Table D-1. Troubleshooting Steps (Continued)

Condition	Possible Problem	Try This:
		6. You may need to use the **cnfg** command (see Appendix A) to change clock speed and/or Ethernet Address, and then later return to **env <Return>** and step 3. 7. Run selftest by typing in **st <Return>** 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 <Return>** 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.
YOU ARE FINISHED (DONE) WITH THIS TROUBLESHOOTING PROCEDURE.		

EIA-232-D Interconnections

E

Introduction

The EIA-232-D standard is the most common 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 E-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 E-1. EIA-232-D Interconnections

Pin Number	Signal Mnemonic	Signal Name and Description
1		CHASSIS GROUND. Not always used. See section *Proper Grounding.*
2	TxD	TRANSMIT DATA. Data to be transmitted; input to the modem from the terminal.
3	RxD	RECEIVE DATA. Data which is demodulated from the receive line; output from the modem to the terminal.
4	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.
5	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.
6	DSR	DATA SET READY. Output from the modem to the terminal to indicate that the modem is ready to transmit data.
7	SIG-GND	SIGNAL GROUND. Common return line for all signals at the modem interface.
8	DCD	DATA CARRIER DETECT. Output from the modem to the terminal to indicate that a valid carrier is being received.
9-14		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 E-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.

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 E-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 E-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 such as `CARRIER NOT PRESENT` to help the user 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 E-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 E-1).

Sample Configurations

Figure E-1 is a good middle-of-the-road 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.

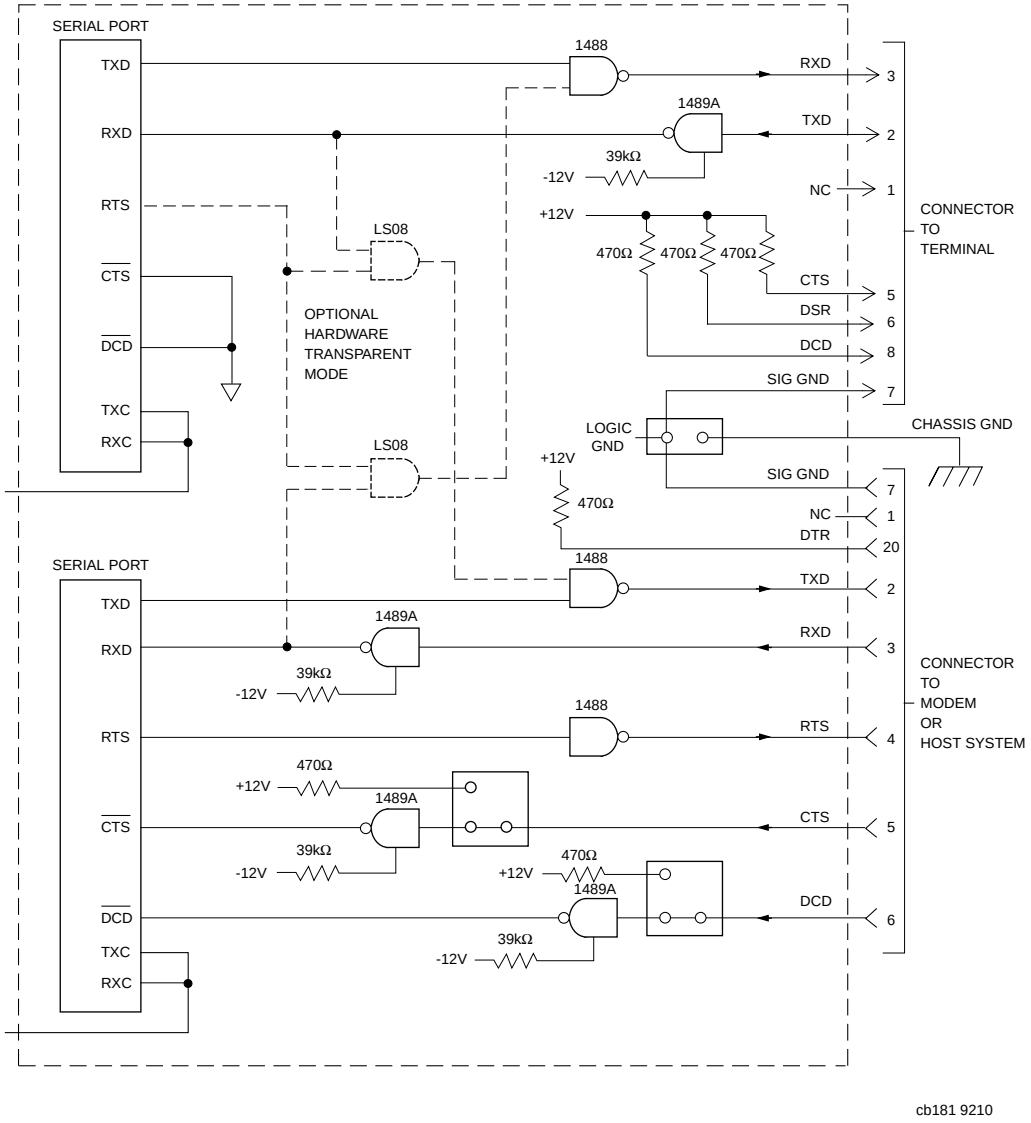


Figure E-1. Middle-of-the-Road EIA-232-D Configuration

Figure E-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 DTR and RTS signals that are 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.

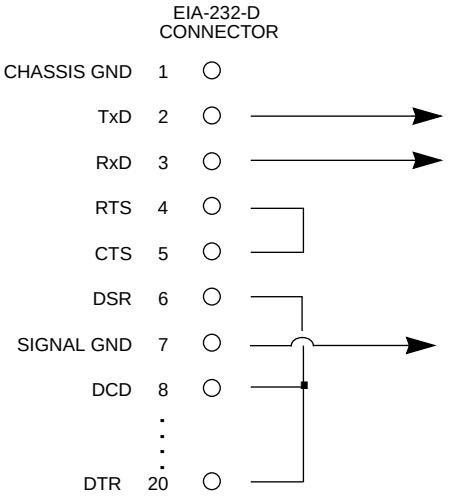


Figure E-2. Minimum EIA-232-D Connection

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 E-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.

Index

C

D

E

F

G

H

I

J

S