MVME166

Single Board Computer

Installation Guide

(MVME166IG/D2)

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 Right Legend

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

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

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

Preface

This manual provides general board level hardware description, hardware preparation and installation instructions, debugger general information, and using the debugger; for the MVME166 Single Board Computer.

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.

A basic knowledge of computers, and digital logic is assumed.

After using this manual, you may wish to become familiar with the publications listed in the *Related Documentation* section in Chapter 1 of this manual. This installation guide is based on these other documents.

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

WARNING

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the documentation for this product, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A Computing Device pursuant to Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user, at the user's own expense, will be required to take whatever measures necessary to correct the interference.

Printed in the United States of America

April 1994

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 either be plugged into an approved three-contact electrical outlet or used with a three-contact to two-contact adapter, with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power outlet. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

Do Not Operate in an Explosive Atmosphere.

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

Keep Away From Live Circuits.

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

Do Not Service or Adjust Alone.

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

Use Caution When Exposing or Handling the CRT.

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

Do Not Substitute Parts or Modify Equipment.

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

Dangerous Procedure Warnings.

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

WARNING

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

Contents

CHAPTER 1 BOARD LEVEL HARDWARE DESCRIPTION

CHAPTER 2 HARDWARE PREPARATION AND INSTALLATION

CHAPTER 3 DEBUGGER GENERAL INFORMATION

CHAPTER 4 USING THE 166Bug DEBUGGER

List of Figures

FIGURES

List of Tables

TABLES

BOARD LEVEL HARDWARE DESCRIPTION 1

Introduction

This chapter describes the board level hardware features of the MVME166 Single Board Computers. The chapter is organized with a board level overview and features list in this introduction, followed by a more detailed hardware functional description. Front panel switches and indicators are included in the detailed hardware functional description. The chapter closes with some general memory maps.

All programmable registers in the MVME166 that reside in ASICs are covered in the *MVME166/MVME167/MVME187 Single Board Computers Programmer's Reference Guide*.

Overview

The MVME166 is based on the MC68040 microprocessor. The MVME166 has 4/8/16/32/64/128/256 MB of ECC-protected DRAM, 1 MB of Flash memory with download EPROM, 128KB of static RAM (with battery backup), 8KB of static RAM and time of day clock (with battery backup), Ethernet transceiver interface, four serial ports with TTL interface, four tick timers, watchdog timer, SCSI bus interface with DMA, Centronics printer port, A16/A24/A32/D8/D16/D32/D64 VMEbus master/slave interface, VMEbus system controller, and a VSB interface.

The I/O connection for the MVME166 is provided by two high density shielded front panel I/O connectors. The SCSI bus is connected through a 68 pin connector. The printer, four serial ports and Ethernet interface are connected through a 100 pin connector. The MVME712-10 transition module and the MVME712-06/07/09 I/O distribution board set were designed to support the MVME166 boards. These transition boards provide configuration headers, serial port drivers and industry standard connectors for the I/O devices.

The VMEbus interface is provided by an ASIC called the VMEchip2. The VMEchip2 includes two tick timers, a watchdog timer, programmable map decoders for the master and slave interfaces, 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, and a VMEbus requester.

Processor-to-VMEbus transfers can be D8, D16, or D32. VMEchip2 DMA transfers to the VMEbus, however, can be D16, D32, D16/BLT, D32/BLT, or D64/MBLT.

The VSBchip2 provides the VSB interface on the MVME166. The VSBchip2 includes programmable map decoders for the master and slave interfaces, a VSB master interface, a VSB slave interface, a VSB interrupter, a VSB interrupt handler, a VSB serial requester, a VSB serial arbiter, and a VSB parallel requester. The VSB is connected to the P2 connector rows A and C on the MVME166.

The PCCchip2 ASIC provides two tick timers and the interface to the LAN chip, SCSI chip, serial port chip, printer port, BBRAM, and download EPROM for Flash memory.

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

Related Documentation

The MVME166 does not ship with all of the documentation that is available for the product. The MVME166 instead ships with a start-up installation guide (the document you are presently reading) that includes all the information necessary to begin working with these products: installation instructions, jumper configuration information, memory maps, debugger/monitor commands, and any other information needed for start-up of the board. The installation guide is MVME166IG/D for the MVME166.

The following publications are applicable to the MVME166 and may provide additional helpful information. They may be purchased by contacting your local Motorola sales office. Non-Motorola documents may be purchased from the sources listed.

Document Title	Motorola Publication Number
MVME166 Single Board Computer User's Manual	MVME166
MVME166 Single Board Computer Support Information	SIMVME166
MVME167Bug Debugging Package User's Manual	MVME167BUG
Debugging Package for Motorola 68K CISC CPUs User's Manual	68KBUG

Document Title	Motorola Publication Number
Single Board Computers SCSI Software User's Manual	SBCSCSI
MVME166 / MVME167 / MVME187 Single Board Computers Programmer's Reference Guide	MVME187PG
MVME712-06 / 07 / 09 I / O Distribution Board Set User's Manual	MVME712IO
MVME712-10 Transition Module User's Manual	MVME712-10
M68040 Microprocessors User's Manual	M68040UM

Notes

The SIMVME166 manual contains: the connector interconnect signal information, parts lists, and the schematics; for the MVME166.

Although not shown in the above list, 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).

These manuals may also be ordered in documentation sets as follows:

68-MVME166SET for use with the MVME166.

MVME166 / D
MVME167BUG / D
68KBUG / D
SBCSCSI / D
MVME187PG / D
SIMVME166 / D

To further assist your development effort, Motorola has collected user's manuals for each of the peripheral controllers used on the MVME166 from the suppliers. This bundle, which can be ordered as part number **68-1X7DS**, includes manuals for the following:

NCR 53C710 SCSI Controller Data Manual and Programmer's Guide
Intel i82596 Ethernet Controller User's Manual
Cirrus Logic CD2401 Serial Controller User's Manual
SGS-Thompson MK48T08 NVRAM/TOD Clock Data Sheet

The following publications are also available from the sources indicated.

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

IEEE Standard for Multiplexed High-Performance Bus Structure: VSB, ANSI/IEEE Std 1096-1988, The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017 (VSB Specification). (This is also *Parallel Sub-system Bus of the IEC 821 VMEbus*, IEC 822 VSB, Bureau Central de la Commission Electrotechnique Internationale; 3,rue de Varembé, Geneva, Switzerland.)

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

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

8*2596CA Local Area Network Coprocessor Data Sheet*, order number 290218; and *82596 User's Manual*, order number 296853; Intel Corporation, Literature Sales, P.O. Box 58130, Santa Clara, CA 95052-8130.

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

MK48T08(B) Timekeeper ™ *and 8Kx8 Zeropower* ™ *RAM* data sheet in *Static RAMs Databook*, order number DBSRAM71; SGS-THOMPSON Microelectronics Group; North & South American Marketing Headquarters, 1000 East Bell Road, Phoenix, AZ 85022-2699.

i28F020 Flash Memory Data Sheet, order number 290245; Intel Literature Sales, P.O. Box 7641, Mt. Prospect, IL 60056-7641.

Requirements

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

- ❑ VMEbus Specification (IEEE 1014-87)
- ❑ EIA-232-D Serial Interface Specification, EIA
- ❑ SCSI Specification, ANSI
- ❑ VSB Specification (IEEE 1096-1988)

Features

Features of the MVME166 are listed below.

- ❑ MC68040 Microprocessor
- ❑ 4/8/16/32/64/128/256MB of 32-bit DRAM with ECC protection
- ❑ 1 MB of Flash memory and a download EPROM
- ❑ 128KB SRAM (with battery backup)
- ❑ Status LEDs for FAIL, STAT, RUN, SCON, LAN, RPWR, SCSI, VME, TPWR and VSB
- ❑ 8K by 8 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 TTL buffers
  - – Centronics printer port
  - – Ethernet transceiver interface with DMA
- ❑ VMEbus interface
  - – VMEbus system controller functions
  - – VMEbus to local bus interface (A24/A32, D8/D16/D32 (D8/D16/D32/D64BLT) (BLT = Block Transfer))
  - – Local bus to VMEbus interface (A16/A24/A32, D8/D16/D32)
  - – VMEbus interrupter
  - – VMEbus interrupt handler
  - – Global CSR for interprocessor communications
  - – DMA for fast local memory - VMEbus transfers (A16/A24/A32, D16/D32 (D16/D32/D64BLT))
- ❑ VSB interface
  - – Local bus to VSB interface (A16/A24/A32, D8/D16/D32)
  - – VSB to local bus interface (A16/A24/A32, D8/D16/D32)
  - – Control and Status Register sets (Board CSRs accessible from both local bus and VSB; local CSRs accessible from local bus) (Includes Global CSR for IPC (General Purpose Registers 1 and 2))
  - – Local bus interrupter
  - – VSB interrupter and VSB interrupt handler
  - – Bidirectional write posting - local bus to VSB and VSB to local bus
  - – EVSB compatible

Specifications

General specifications for the MVME166 are listed in Table 1-1.

Table 1-1. MVME166 Specifications

Characteristics	Specifications
Power requirements (excluding external LAN transceiver) (at 33 MHz with 32 MB ECC memory)	+5 Vdc (± 5%), 5.0 A (typical), 6.5 A (max.) (includes transition modules) +12 Vdc (± 5%), 100 mA (max.) (1.0 A (max.) with offboard LAN transceiver) -12 Vdc (± 5%), 100 mA (max.)
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 PC board with mezzanine module only Height Depth Thickness PC boards with connectors and front panel Height Depth Thickness	Double-high VMEboard 9.187 inches (233.35 mm) 6.299 inches (160.00 mm) 0.662 inches (16.77 mm) 10.309 inches (261.85 mm) 7.4 inches (188 mm) 0.80 inches (20.32 mm)

Manual Terminology

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

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

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

Unless otherwise specified, all address references are in hexadecimal.

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

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

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

Data and address sizes are defined as follows:

- A byte is eight bits, numbered 0 through 7, with bit 0 being the least significant.
- A two-byte is 16 bits, numbered 0 through 15, with bit 0 being the least significant. For the MVME166 and other CISC modules, this is called a word.
- A four-byte is 32 bits, numbered 0 through 31, with bit 0 being the least significant. For the MVME166 and other CISC modules, this is called a longword.

The terms control bit and status bit are used extensively in this document. The term control bit is used to describe a bit in a register that can be set and cleared under software control. The term true is used to indicate that a bit is in the state that enables the function it controls. The term false is used to indicate that the bit is in the state that disables the function it controls. 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. The term status bit is used to describe a bit in a register that reflects a specific condition. The status bit can be read by software to determine operational or exception conditions.

Block Diagram

Figure 1-1 is a general block diagram of the MVME166.

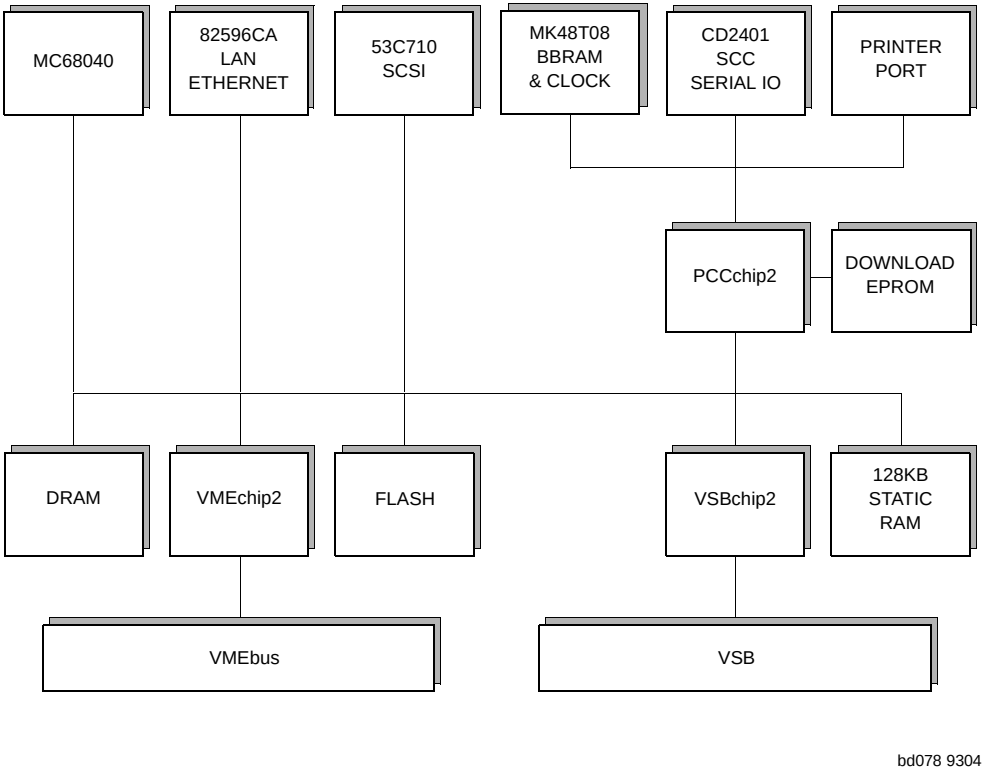


Figure 1-1. MVME166 Block Diagram

Functional Description

This section contains a functional description of the major blocks on the MVME166 Single Board Computers.

Front Panel Switches and Indicators

There are switches and LEDs on the front panel of the MVME166. The switches are RESET and ABORT. The RESET switch resets all onboard devices and drives SYSRESET* if the board is system controller. The RESET switch may be disabled by software.

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.

There are ten LEDs on the MVME166 front panel: FAIL, STAT, RUN, SCON, LAN, RPWR, SCSI, VME, TPWR and VSB.

The red FAIL LED (part of DS1) lights when the BRDFAIL signal line is active.

The MC68040 status lines are decoded, on the MVME166, to drive the yellow STAT (status) LED (part of DS1). In this case, a halt condition from the processor lights the LED.

The green RUN LED (part of DS2) lights when the local bus TIP* signal line is low. This indicates one of the local bus masters is executing a local bus cycle.

The green SCON LED (part of DS2) lights when the VMEchip2 in the MVME166 is the VMEbus system controller.

The green LAN LED (part of DS3) lights when the LAN chip is local bus master.

The MVME166 supplies +5V, +12V, and -12V power to the transition board through fuses. There is one fuse for each voltage. The green RPWR (remote power) LED (part of DS3) lights when all three voltages are available to the transition board interface.

The green SCSI LED (part of DS4) lights when the SCSI chip is local bus master.

The green VME LED (part of DS4) lights when the board is using the VMEbus (VMEbus AS* is asserted by the VMEchip2) or when the board is accessed by the VMEbus (VMEchip2 is the local bus master).

The MVME166 supplies +5V to the SCSI bus for terminator power through a fuse. The green TPWR (terminator power) LED (part of DS5) lights when TERMPWR is available to the SCSI bus. SCSI bus TERMPWR may be supplied by other devices on the SCSI bus.

The green VSB LED (part of DS5) lights when the MVME166 is using the VSB (VSB PAS* is asserted by the VSBchip2) or when the MVME166 is accessed by the VSB (VSBchip2 is the local bus master).

Data Bus Structure

The local data bus on the MVME166 is a 32-bit synchronous bus that is based on the MC68040 bus, and supports burst transfers and snooping. The various local bus master and slave devices use the local bus to communicate. The local bus is arbitrated by priority type arbiter and the priority of the local bus masters from highest to lowest is: 82596CA LAN, CD2401 serial (through the PCCchip2), 53C710 SCSI, VSB, VMEbus, and MPU. In the general case, any master can access any slave; however, not all combinations pass the common sense test. Refer to the *MVME166/MVME167/MVME187 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 MVME166. The MC68040 has on-chip instruction and data caches and a floating point processor. Refer to the M68040 user's manual for more information.

Flash Memory and Download EPROM

The MVME166 includes four 28F020 Flash memory devices and a download EPROM. These parts replace the four EPROM sockets used on the MVME167/187. The Flash parts are programmable on the MVME166 board and the programming code is provided in the download EPROM. The Flash devices provide 1 MB of ROM at address $FF800000-$FF8FFFFF. The download EPROM provides 128 KB of ROM at $FFF80000-$FFF9FFFF. The download EPROM is 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. The download EPROM appears at 0 until the DR0 bit is cleared in the PCCchip2 chip. The Flash devices are controlled by the VMEchip2 and the download EPROM is controlled by the PCCchip2. The PC0 bit in the MC68230 PI/T chip must be low to enable writes to Flash.

The EPROM contains the BootBug product (166BBug). Because Flash memory can be electronically erased, the EPROM firmware is a subset of the regular debugger product. It contains enough functionality from the debugger to permit downloading of object code (via VMEbus, serial port, SCSI bus, or the network) and reprogramming of the Flash memory.

A jumper on the MVME166 (J3, pins 7 and 8) controls the operation of the BootBug. If the jumper is in place, the BootBug (which always executes at power-up and reset) passes execution to the full debugger contained in Flash memory. If the jumper is removed, execution continues (with diminished functionality) in the BootBug.

Before you perform any SCSI, VMEbus, or Ethernet I/O with the MVME166, it may be necessary to define some parameters (e.g., SCSI ID, Ethernet address, VMEbus mapping). For details on configuring the MVME166, refer to the **setup** command description in Chapter 3 in this manual, and in the *MVME167Bug Debugging Package User's Manual*.

SRAM

The boards include 128KB of 32-bit wide static RAM with onboard battery backup that supports 8-, 16-, and 32-bit wide accesses. The SRAM allows the debugger to operate and limited diagnostics to be executed without the DRAM mezzanine. The SRAM is controlled by the VMEchip2, and the access time is programmable. The boards are populated with 100 ns SRAMs.

The SRAM is also battery backed up on the MVME166. The battery backup function is provided by a Dallas DS1210S. The DS1210S supports primary and secondary power sources. When the main board power fails, the DS1210S selects the source with the highest voltage. If one source should fail, the DS1210S switches to the redundant source. Each time the board is powered, the DS1210S checks power sources and if the voltage of the backup sources is less than two volts, the second memory cycle is blocked. This allows software to provide an early warning to avoid data loss. Because the DS1210S may block the second access, the software should do at least two accesses before relying on the data.

The MVME166 provides jumpers that allow either power source of the DS1210S to be connected to the VMEbus +5 V STDBY pin or one cell of the onboard battery. For example, the primary system backup source may be a battery connected to the VMEbus +5 V STDBY pin and the secondary source may be the onboard battery. If the system source should fail or the board is removed from the chassis, the onboard battery takes over.

Caution **For proper operation of the SRAM, some jumper combination must be installed on the Backup Power Source Select Header. If one of the jumpers is used to select the battery, the battery must be installed on the MVME166. The SRAM may malfunction if inputs to the DS1210S are left unconnected.**

The onboard power source is a RAYOVAC FB1225 battery which has two BR1225 type lithium cells and is socketed for easy removal and replacement. A small capacitor is provided to allow the battery to be quickly replaced without data loss. The lifetime of the battery is very dependent on the ambient temperature of the board and the power-on duty cycle. The lithium battery supplied on the MVME166 should provide at least two years of backup time

with the board powered off and the board at 40° C. If the power-on duty cycle is 50% (the board is powered on half of the time), the battery lifetime is four years. At lower ambient temperatures the backup time is greatly extended and may approach the shelf life of the battery. When a board is stored, the battery should be disconnected to prolong battery life. This is especially important at high ambient temperatures. The MVME166 is shipped with the batteries disconnected.

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

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, or new and old batteries together.
- ❏ Do not charge.
- ❏ Always check proper polarity.

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

Before installing a new battery, ensure that the battery pins are clean. Note the battery polarity and press the battery into the socket. When the battery is in the socket, no soldering is required.

Onboard DRAM

The MVME166 onboard DRAM is located on a mezzanine board. The mezzanine boards are available in different sizes and with ECC protection. Mezzanine board sizes are 4, 8, 16, 32, 64, or 128MB, and two mezzanine boards may be stacked to provide 256MB of onboard RAM. The main board and a single mezzanine board together take one slot. The stacked configuration requires two VMEboard slots. The DRAM is four-way interleaved to efficiently support cache burst cycles.

The DRAM map decoder can be programmed to accommodate different base address(es) and sizes of mezzanine boards. The onboard DRAM is disabled by a local bus reset and must be programmed before the DRAM can be accessed. Refer to the MCECC in the *MVME166/MVME167/MVME187 Single Board Computers Programmer's Reference Guide* for detailed programming information. Most DRAM devices require some number of access cycles before the DRAMs are fully operational. Normally this requirement is met by the onboard refresh circuitry and normal DRAM installation. However, software should insure a minimum of 10 initialization cycles are performed to each bank of RAM.

Battery Backed Up RAM and Clock

The MK48T08 RAM and clock chip is used on the MVME166. This chip provides a time of day clock, oscillator, crystal, power fail detection, memory write protection, 8KB of RAM, and a battery in one 28-pin package. The clock provides seconds, minutes, hours, day, date, month, and year in BCD 24-hour format. Corrections for 28-, 29- (leap year), and 30-day months are automatically made. No interrupts are generated by the clock. The 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 local bus to VMEbus interface, the VMEbus to local bus interface, and the local-VMEbus DMA controller functions on the MVME166 are provided by the VMEchip2. The VMEchip2 can also provide the VMEbus system controller functions.

VME Subsystem Bus (VSB) Interface

The local bus to VSB interface and the VSB to local bus interface are provided by the VSBchip2, only on the MVME166 board. The VSB uses the P2 connector of the MVME166.

I/O Interfaces

The MVME166 provides onboard I/O for many system applications. The I/O functions include serial ports, printer port, Ethernet transceiver interface, and SCSI mass storage interface.

Serial Port Interface

The CD2401 serial controller chip (SCC) is used to implement the four serial ports. The serial ports support the standard baud rates (110 to 38.4K baud). The four serial ports on the MVME166 are functionally the same. All serial ports are full function asynchronous or synchronous ports. They can operate

at synchronous bit rates up to 64 k bits per second. They use RXD, CTS, DCD, TXD, RTS, DTR, and DSR. They also interface to the synchronous clock signal lines. Additional control signals are provided for each serial port by the MC68230. These include local loopback control, self test control, and ring indicator. The ring indicator signal can be programmed to generate a local bus interrupt. Refer to the MC68230 section for additional information. Note that the usable functionality of the serial ports depends on the transition module used.

All four serial ports on the MVME166 use a TTL interface to the transition board. This allows the interface specific drivers to be located on the transition board. This allows more flexibility in configuring the serial ports for different interfaces like EIA-232-D or V.35. An external I/O transition module such as the MVME712-10 should be used to provide configuration headers, interface drivers, and industry-standard connectors.

The interface provided by the PCCchip2 allows the 16-bit CD2401 to appear at contiguous addresses; however, accesses to the CD2401 must be 8 or 16 bits. 32-bit accesses are not permitted. Refer to the CD2401 data sheet for detailed programming information.

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

MC68230 Parallel Interface/Timer

The MVME166 provides an MC68230 parallel interface/timer (PI/T) chip. When the MVME166 is used with the MVME712-10 transition module or the MVME712-06/07/09 I/O distribution board set, the MC68230 is used to provide additional control lines for the serial ports. These include local loopback, self test, and ring indicator. The ring indicator signals can be programmed to generate local bus interrupts. Refer to the MVME712-10 transition module manual for more information.

The base address of the MC68230 is $FFF45E00, and because it is an 8-bit device it appears only at odd addresses. Space for the MC68230 was created by dividing the area occupied by redundant copies of the CD2401 registers into eight segments. The CD2401 is still addressed at $FFF45000 to $FFF451FF. Addresses $FFF45200 to $FFF45BFF are reserved, and if accessed on an MVME166 cause a local bus timeout error, if the local bus timer is enabled. The address range from $FFF45C00 to $FFF45DFF always returns a local bus timeout error if the local bus timer is enabled. The CD2401 appears redundantly from $FFF45200 to $FFF45FFF on the MVME167/187.

The presence of the MC68230 can be determined by reading address $FFF45C00. If a timeout error occurs, then the board is an MVME166 and and the MC68230 is present. If a timeout does not occur, then the board is an MVME167/187 and the MC68230 is not present. The local bus timeout timer in the VMEchip2 must be enabled for this test.

The MC68230 may be used for general purpose I/O when the MVME166 is not used with the MVME712 family of transition modules. Because the outputs are unbuffered and unprotected, these signals should be used with caution. The port A signal lines PA<7..0> are connected to the front panel connector J9. The port A signal lines can be programmed as inputs or outputs. The port B signal lines PB<3..0> are connected to the port H signal lines H<4..1> and the front panel connector J9. This allows these four lines to be inputs or outputs or receive interrupts. The port B signal line PB<7> is also connected to the front panel connector J9. When used with the MVME712 family of transition modules, the PB<7> signal line is used to read the configuration of the serial ports. Timer interrupts from the MC68230 are not supported on the MVME166. The MC68230 is connected to a 10 MHz clock. The PC0 bit in the MC68230 PI/T chip must be low to enable writes to Flash memory.

Parallel Port Interface

The PCCchip2 provides an 8-bit bidirectional parallel port. All eight bits of the port must be either inputs or outputs (no individual selection). In addition to the 8 bits of data, there are two control pins and five status pins. Each of the status pins can generate an interrupt to the MPU in any of the following programmable conditions: high level, low level, high-to-low transition, or low-to-high transition. This port may be used as a Centronics-compatible parallel printer port or as a general parallel I/O port.

When used as a parallel printer port, the five status pins function as: Printer Acknowledge (ACK), Printer Fault (FAULT*), Printer Busy (BSY), Printer Select (SELECT), and Printer Paper Error (PE); while the control pins act as Printer Strobe (STROBE*), and Input Prime (INP*).

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

Ethernet Interface

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

Every MVME166 is assigned an Ethernet Station Address. The address is $08003E2XXXXX where XXXXX is the unique 5-nibble number assigned to the board (i.e., every MVME166 has a different value for XXXXX).

Each module has an Ethernet Station Address displayed on a label attached to the VMEbus P2 connector. In addition, the six bytes including the Ethernet address are stored in the configuration area of the BBRAM. That is, 08003E2XXXXX is stored in the BBRAM. At an address of $FFFC1F2C, the upper four bytes (08003E2X) can be read. At an address of $FFFC1F30, the lower two bytes (XXXX) can be read. The MVME166 debugger has the capability to retrieve or set the Ethernet address. So does the MVME166 BootBug.

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

The Ethernet transceiver interface is located on the MVME166 main module, and the industry standard connector is located on the MVME712X transition module.

Support functions for the 82596CA are provided by the PCCchip2. Refer to the 82596CA user's guide for detailed programming information.

SCSI Interface

The MVME166 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

The individual configuring the system must ensure that the SCSI bus is properly terminated at both ends. On the MVME166, the SCSI bus termination is provided on the main board. The terminators are enabled/disabled by a jumper. If the SCSI bus ends at the MVME166, the SCSI terminators must be enabled by installing the jumper. Refer to the jumper configuration tables in Chapter 2.

Local Resources

The MVME166 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 MVME166 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 or VSB bound cycles. VMEbus bound cycles are timed by the VMEbus access timer and the VMEbus global timer. VSB bound cycles are timed by the VSB access timer, the VSB transfer timer, and if its serial arbiter is enabled, by the VSB arbitration timer.

Connectors

The MVME166 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, on the MVME166, provide the connection to the VSB. The MVME166 has a 20-pin connector mounted behind the front panel. When the MVME166 board is enclosed in a chassis and the front panel is not visible, this connector allows the reset, abort and LED functions to be extended to the control panel of the system, where they are visible. The MVME166 has a 68-pin mini D ribbon shielded connector for the SCSI bus interface. The MVME166 has a 100-pin mini D ribbon shielded connector for the serial ports, Ethernet, and printer.

Memory Maps

There are two points of view for memory maps: 1) the mapping of all resources as viewed by local bus masters (local bus memory map), 2) the mapping of onboard resources as viewed by externa masters (VMEbus memory map or VSB memory map).

Local Bus Memory Map

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

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 MVME166, Transfer Types 0, 1, and 2 define the normal address range.

Table 1-2 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 1-3 further defines the map for the local I/O devices.

Table 1-2. 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 or VSB)	D32 / D16	3GB	?	3, 4
$FF800000 - $FF8FFFFF	FLASH	D32	1MB	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. There is 1MB of FLASH in this 4MB map area. Download EPROM on the MVME166 appears at $00000000 - ROMSIZE following a local bus reset. The Download EPROM appears at 0 until the DR0 bit is cleared in the PCCchip2. The DR0 bit is located at address $FFF42000 bit 15. The EPROM must be disabled at 0 before the DRAM is enabled. The VMEchip2, VSBchip2, 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 MCECC chip, and the local-to-VMEbus decoders are programmed in the VMEchip2. The local-to-VSB decoders are programmed in the VSBchip2.
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 1-3. Local I/O Devices Memory Map

Address Range	Devices Accessed	Port Size	Size	Notes
$FFF00000 - $FFF3FFFF	reserved	--	256KB	5
$FFF40000 - $FFF400FF	VMEchip2 (LCSR)	D32	256B	1,4
$FFF40100 - $FFF401FF	VMEchip2 (GCSR)	D32-D8	256B	1,4
$FFF40200 - $FFF40FFF	reserved	--	3.5KB	5,7
$FFF41000 - $FFF41FFF	VSBchip2	D32-D8	4KB	1,10
$FFF42000 - $FFF42FFF	PCCchip2	D32-D8	4KB	1
$FFF43000 - $FFF430FF	MCECC #1	D8	256B	1
$FFF43100 - $FFF431FF	MCECC #2	D8	256B	1
$FFF43200 - $FFF43FFF	MCECCs (repeated)	--	3.5KB	1,7
$FFF44000 - $FFF44FFF	reserved	--	4KB	5
$FFF45000 - $FFF451FF	CD2401 (Serial Comm. Cont.)	D16-D8	512B	1,9
$FFF45200 - $FFF45DFF	reserved	--	3KB	7,9
$FFF45E00 - $FFF45FFF	MC68230	--	512B	1,9
$FFF46000 - $FFF46FFF	82596CA (LAN)	D32	4KB	1,8
$FFF47000 - $FFF47FFF	53C710 (SCSI)	D32 / D8	4KB	1
$FFF48000 - $FFF4FFFF	reserved	--	32KB	5
$FFF50000 - $FFF6FFFF	reserved	--	128KB	5
$FFF70000 - $FFF76FFF	reserved	--	28KB	6
$FFF77000 - $FFF77FFF	reserved	--	4KB	2
$FFF78000 - $FFF7EFFF	reserved	--	28KB	6
$FFF7F000 - $FFF7FFFF	reserved	--	4KB	2
$FFF80000 - $FFF9FFFF	Download EPROM	--	128KB	11
$FFFA0000 - $FFFBFFFF	reserved	--	128KB	5
$FFFC0000 - $FFFCFFFF	MK48T08 (BBRAM, TOD Clock)	D32-D8	64KB	1
$FFFD0000 - $FFFDFFFF	reserved	--	64KB	5
$FFFE0000 - $FFFEFFFF	reserved	--	64KB	2

NOTES:

1. For a complete description of the register bits, refer to the *MVME166/MVME167/MVME187 Single Board Computers Programmer's Reference Guide* or to the data sheet for the specific chip.

2. On the MVME166 this area does not return an acknowledge signal. If the local bus timer on the MVME166 is enabled, the access times out and is terminated by a TEA signal.

   On the MVME187 this area is used.

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

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

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

6. This area does return an acknowledge signal.

7. Size is approximate.

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

9. The MC68230 is included only on the MVME166. The area from $FFF45200 to $FFF45DFF does not return an acknowledge on the MVME166. If the local bus timer is enabled, the access times out and is terminated by a TEA signal.

10. The VSBchip2 is included only on the MVME166.

11. The Download EPROM is only on the MVME166.

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-programmable 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 MVME166 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.

VSB Memory Map

This section describes the mapping of local resources as viewed by VSB masters. The VSBchip2, on the MVME166, includes a user-programmable map decoder for the VSB to local bus interface. This map decoder allows VSB masters access to devices on the local bus. Default addresses for the slave, master, and address decoders are provided by the **ENV** command. Refer to Appendix A.

HARDWARE PREPARATION AND INSTALLATION 2

Introduction

This chapter provides unpacking instructions, hardware preparation, and installation instructions for the MVME166. Hardware preparation and installation for the MVME712 series transition modules is described in a separate manual. Refer to the *Related Documentation* section in Chapter 1.

Unpacking Instructions

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

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

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

Hardware Preparation

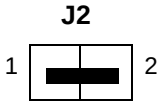
To select the desired configuration and ensure proper operation of the MVME166, certain option modifications may be necessary before installation. The MVME166 provides software control for most of these options. Some options can not be done in software, so are done by jumpers on headers. Most other modifications are done by setting bits in control registers after the MVME166 has been installed in a system. (The MVME166 registers are described in the *MVME166/MVME167/MVME187 Single Board Computers Programmer's Reference Guide* as listed in *Related Documentation* in Chapter 1.)

Figure 2-1 illustrates the placement of the switches, jumper headers, connectors, and LED indicators on the MVME166. The MVME166 has been factory tested and is shipped with the factory jumper settings described in the following sections. The MVME166 operates with its required and factory-installed Debug Monitor, MVME166Bug (166Bug), with these factory jumper settings. Settings can be changed for the following headers:

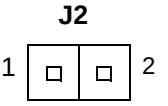
- ❏ SCSI terminator selection (J2)
- ❏ General purpose readable register (J3)
- ❏ System controller selection (J6)
- ❏ SRAM backup power source selection (J7)

SCSI Terminator Enable Header J2

The MVME166 provides terminators for the SCSI bus. The SCSI terminators are enabled / disabled by jumpers on header J2. The SCSI terminators may be configured as follows.



Onboard SCSI Bus Terminator Enabled (Factory Configuration)



Onboard SCSI Bus Terminator Disabled

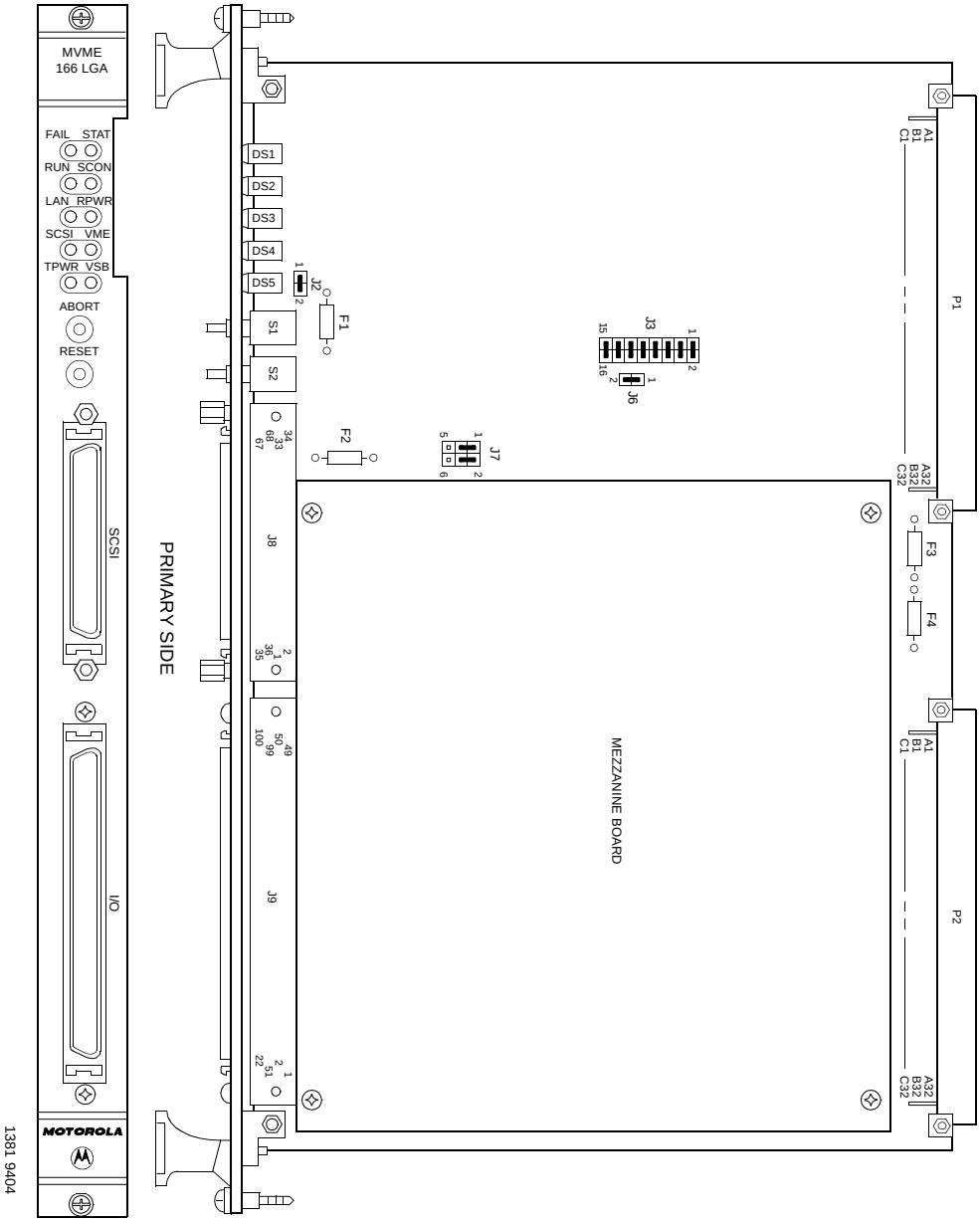
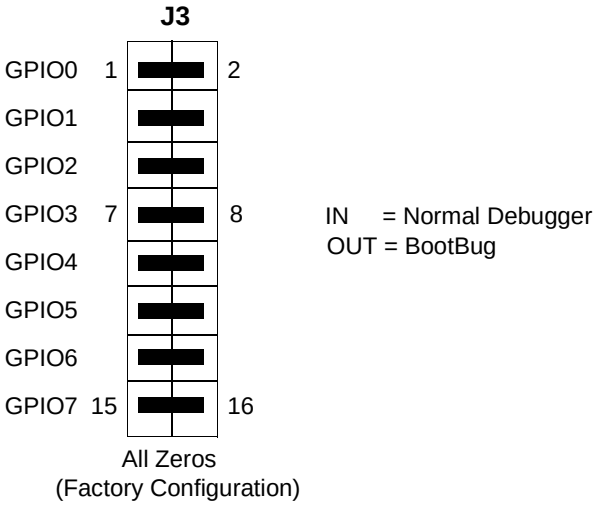


Figure 2-1. MVME166 Switches, Headers, Connectors, Fuses, and LEDs

General Purpose Readable Jumpers on Header J3

Each MVME166 may be configured with readable jumpers. These jumpers can be read as a register (at $FFF40088) in the VMEchip2 LCSR. The bit values are read as a one when the jumper is off, and as a zero when the jumper is on.

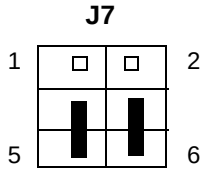


System Controller Header J6

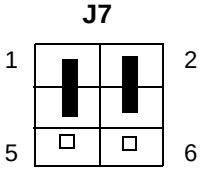
The MVME166 can operate as VMEbus system controller. The system controller function is enabled / disabled by jumpers on header J6. When the MVME166 is functioning as system controller, the SCON LED is turned on. The VMEchip2 can be configured as a system controller as follows.

SRAM Backup Power Source Select Header J7

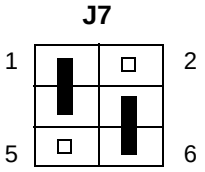
Header J7 is used to select the power source used to backup the SRAM on the MVME166.



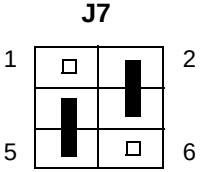
Primary Source OnboardBattery
Secondary Source Onboard Battery



Primary Source VMEbus +5V STBY
Secondary Source VMEbus +5V STBY
(Factory Configuration)



Primary Source VMEbus +5V STBY
Secondary Source Onboard Battery



Primary Source Onboard Battery
Secondary Source VMEbus +5V STBY

Caution **Do not remove all jumpers from J7. This may disable the SRAM.**
If the battery is removed, jumpers must be installed on J7, between pins 1 to 3 and pins 2 to 4, as shown in the Factory Configuration drawing above.

2

Installation Instructions

The following sections discuss the installation of the MVME166 in a VME chassis, and describe system considerations relevant to the installation. Ensure that an EPROM device is installed as needed. The factory configuration provides for one EPROM (installed for 166BBug, the BootBug firmware subset of the MVME166Bug debug monitor contained in Flash memory) in socket U12. Ensure that all header jumpers are configured as desired.

MVME166 Module Installation

Now that the MVME166 module is ready for installation, proceed as follows:

a. Turn all equipment power OFF and disconnect power cable from ac power source.

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

WARNING **DANGEROUS VOLTAGES, CAPABLE OF CAUSING DEATH, ARE PRESENT IN THIS EQUIPMENT. USE EXTREME CAUTION WHEN HANDLING, TESTING, AND ADJUSTING.**

b. Remove chassis cover as instructed in the equipment user's manual.

c. Remove the filler panel(s) from the appropriate card slot(s) at the front and (if the chassis has a rear card cage that you intend to use) rear of the chassis.

The MVME166 module requires power from both P1 and P2. It may be installed in any double-height unused card slot, if it is not configured as system controller. If the MVME166 is configured as system controller, it must be installed in the leftmost card slot (slot 1) to correctly initiate the bus-grant daisy-chain and to have proper operation of the IACK-daisy-chain driver.

The MVME166 is to be installed in the front of the chassis. The MVME712 module(s) is / are to be installed either in the front or the rear of the chassis, depending on the I/O option you select:

Through the front panel I/O connector via cable to the MVME712-10 transition module, which may be mounted either in the forward card cage alongside the MVME166 (recommended) or in the rear transition module area.

Through the front panel I/O and SCSI connectors via cable to the MVME712-06/07/09 I/O distribution board set, which likewise may be mounted either in the forward card cage alongside the MVME166 or in the rear transition module area.

If you opt for front mounting of the transition module(s), the MVME166 should be located at their left if possible to make use of the cabling slots provided in its front panel. You may need to shift the placement of other modules in the forward card cage to allow space for the double- or triple-wide MVME166/MVME712 combination.

d. Carefully slide the MVME166 module into the card slot. Be sure the module is seated properly in the P1 and P2 connectors on the backplane. Do not damage or bend connector pins. Fasten the module in the chassis with the screws provided, making good contact with the transverse mounting rails to minimize RFI emissions.

e. On the chassis backplane, remove the IACK and BG jumpers from the header for the card slot occupied by the MVME166.

f. Connect the transition module(s) and specified cable(s) to the MVME166 (at the I/O connector on the MVME166 front panel, according to configuration) to mate with (optional) terminals or other peripherals at the serial ports, parallel port, SCSI ports, and LAN Ethernet port. Connect the peripherals to the cable(s).

Refer to the manuals listed in *Related Documentation* in Chapter 1 for information on installing the MVME712 series transition modules. (Some connection diagrams are in the *MVME166/MVME167/MVME187 Single Board Computers Programmer's Reference Guide*.) Some cables you require may not be provided with the MVME712 series modules; you may need to fabricate or otherwise provide those cables. (Motorola recommends using shielded

cables for all connections to peripherals to minimize radiation.) Connect the peripherals to the cable(s).

g. Install any other required VMEmodules in the system.

h. Replace the chassis cover.

i. Connect the power cable to the ac power source and turn the equipment power ON.

System Considerations

The MVME166 draws power from both the P1 and P2 connectors on 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 MVME166 may not operate properly without its main board connected to P1 and P2 of the VMEbus backplane.

Whether the MVME166 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 *MVME166/MVME167/MVME187 Single Board Computers Programmer's Reference Guide* as listed in *Related Documentation* in Chapter 1.

The VME Subsystem Bus (VSB, a subset of the VMEbus) used in the MVME166 is a local extension bus. It allows the MVME166 to access additional memory and I/O over a local bus, removing traffic from the global VMEbus and improving the total throughput of the system. The VSB interface occupies 64 I/O pins on connector P2 and utilizes the multiplexing of address and data in order to achieve full 32-bit functionality, along with appropriate control signals, within the 64-pin allotment.

The MVME166 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 MVME166Bug firmware. This may be changed, by software, to any other base address. Refer to the *MVME166/MVME167/MVME187 Single Board Computers Programmer's Reference Guide* for details.

If the MVME166 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 MVME166 waits forever for the VMEbus cycle to complete. This would cause the system to hang up. There is only one situation in which the system might lack this global bus timeout: the MVME166 is not the system controller, and there is no global bus timeout elsewhere in the system.

Multiple MVME166 modules may be configured into a single VME card cage. In general, hardware multiprocessor features are supported.

Other MPUs on the VMEbus can interrupt, disable, communicate with and determine the operational status of the processor(s). One register of the GCSR set includes four bits which function as location monitors to allow one MVME166 processor to broadcast a signal to other MVME166 processors, if any are present. All eight registers are accessible from any local processor as well as from the VMEbus.

The MVME166 provides +12 Vdc, -12 Vdc, and +5 Vdc power to the transition modules through fuses F1, F3, and F4. The fused +5 Vdc power is also provided to the 20-pin remote reset connector. These voltage sources are used by the transition modules to power the serial port drivers and any LAN transceivers connected to the transition module. The RPWR LED (DS3) on the MVME166 front panel lights when all three voltages are available.

The MVME166 provides +5 Vdc to the SCSI bus TERMPWR signal through fuse F2, located near the front panel SCSI bus connector. The TPWR LED (DS5) on the MVME166 front panel monitors the SCSI bus TERMPWR signal; when the MVME166 is connected to an SCSI bus, either directly or via the MVME712-07 module, the TPWR LED lights when there is SCSI terminator power. Because any device on the SCSI bus can provide TERMPWR, the LED does not directly indicate the condition of the fuse. If the LED is not illuminated during SCSI bus operation, the fuse should be checked.

DEBUGGER GENERAL INFORMATION 3

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 MVME166 Single Board Computer, and is known as the MVME166Bug, or just 166Bug.

Description of 166Bug

The 166Bug package, MVME166Bug, is a powerful evaluation and debugging tool for systems built around the MVME166 CISC-based microcomputers. Facilities are available for loading and executing user programs under complete operator control for system evaluation. 166Bug includes commands for display and modification of memory, breakpoint and tracing capabilities, a powerful assembler / disassembler useful for patching programs, and a self-test at power-up feature which verifies the integrity of the system. Various 166Bug routines that handle I / O, data conversion, and string functions are available to user programs through the TRAP #15 system calls.

166Bug consists of three parts:

- A command-driven user-interactive software debugger, described in Chapter 4 and hereafter referred to as "the debugger" or "166Bug".
- A command-driven diagnostic package for the MVME166 hardware, hereafter referred to as "the diagnostics".
- A user interface which accepts commands from the system console terminal.

When using 166Bug, you operate out of either the debugger directory or the diagnostic directory. If you are in the debugger directory, the debugger prompt "`166-Bug>`" is displayed and you have all of the debugger commands at your disposal. If you are in the diagnostic directory, the diagnostic prompt "`166-Diag>`" is displayed and you have all of the diagnostic commands at your disposal as well as all of the debugger commands. You may switch between directories by using the Switch Directories (**SD**) command, or may examine the commands in the particular directory that you are currently in by using the Help (**HE**) command.

Because 166Bug is command-driven, it performs its various operations in response to user commands entered at the keyboard. When you enter a command, 166Bug executes the command and the prompt reappears. However, if you enter a command that causes execution of user target code (e.g., "**GO**"), then control may or may not return to 166Bug, depending on the outcome of the user program.

If you have used one or more of Motorola's other debugging packages, you will find the CISC 166Bug 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.

166Bug Implementation

MVME166Bug 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, 166Bug is contained in four Flash memory components. The onboard Flash memory provides 1.0MB (256KB longwords) of nonvolatile storage. The 166Bug consumes the first half (512KB) of this memory, leaving the second half available for user applications. A command is provided, both in the regular "Bug" product and the "BootBug" product, to allow erasing and reprogramming this Flash memory.

WARNING

Reprogramming any portion of Flash memory, will erase everything currently contained in Flash, including the 166Bug product! You must copy the 166Bug from Flash to RAM, combine your application with the 166Bug image, and then reprogram Flash with the combined object image.

Installation and Startup

Even though the MVME166Bug firmware is installed on the MVME166 module, for 166Bug to operate properly with the MVME166, follow this set-up procedure.

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

1. Turn all equipment power OFF. Refer to the *Hardware Preparation* section in Chapter 2 and install / remove jumpers on headers as required for your particular application. Jumpers on header J3 affect 166Bug operation as listed below. The default condition is with all eight jumpers installed, between pins 1-2, 3-4, 5-6, 7-8, 9-10, 11-12, 13-14, and 15-16.

   The MVME166 may be configured with these readable jumpers. These jumpers can be read as a register (at $FFF40088) in the VMEchip2 LCSR. The bit values are read as a one when the jumper is off, and as a zero when the jumper is on. This jumper block (header J3) contains eight bits. Refer to the *MVME166/MVME167/MVME187 Single Board Computers Programmer's Reference Guide*.

   The MVME166Bug reserves / defines the four lower order bits (GPI3 to GPI0). The following is the description for the bits reserved / defined by the debugger:

Bit	J3 Pins	Description
Bit #0 (GPI0)	1-2	When this bit is a one (high), it instructs the debugger to use local Static RAM for its work page (i.e., variables, stack, vector tables, etc.).
Bit #1 (GPI1)	3-4	When this bit is a one (high), it instructs the debugger to use the default setup / operation parameters in ROM versus the user setup / operation parameters in NVRAM. This is the same as depressing the RESET and ABORT switches at the same time. This feature can be used in the event the user setup is corrupted or does not meet a sanity check. Refer to the **ENV** command (Appendix A) for the ROM defaults.
Bit #2 (GPI2)	5-6	Reserved for future use.
Bit #3 (GPI3)	7-8	When this bit is a one (jumper out) the BootBug will continue execution after reset or power up. Normal operation (jumper in) results in the BootBug executing the **166Bug** debugger in **Flash** memory.
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.

Note that when the MVME166 comes up in a cold reset, 166Bug runs in System Mode. Using the Environment (**ENV**) or **MENU** commands can make 166Bug run in Board mode. Refer to Appendix A.

2. Configure header J6 by installing / removing a jumper between pins 1 and 2. A jumper installed / removed enables / disables the system controller function of the MVME166.

3. Refer to the set-up procedure for your particular chassis or system for details concerning the installation of the MVME166.
4. Connect the terminal which is to be used as the 166Bug system console to the default debug EIA-232-D port at serial port 1 on front panel I/O connector J9 through an MVME712-10 or MVME712-06 transition module. Refer to the *MVME166/MVME167/MVME187 Single Board Computers Programmer's Reference Guide* for some possible connection diagrams. Set up the terminal as follows:
   - eight bits per character
   - one stop bit per character
   - parity disabled (no parity)
   - baud rate 9600 baud (default baud rate of MVME166 ports at power-up)

   After power-up, the baud rate of the debug port can be reconfigured by using the Port Format (**PF**) command of the 166Bug debugger.

Note **In order for high-baud rate serial communication between 166Bug and the terminal to work, the terminal must do some form of handshaking. If the terminal being used does not do hardware handshaking via the CTS line, then it must do XON/XOFF handshaking. If you get garbled messages and missing characters, then you should check the terminal to make sure XON/XOFF handshaking is enabled.**

5. If you want to connect devices (such as a host computer system and/or a serial printer) to the other EIA-232-D port connectors (marked SERIAL PORTS 2, 3, and 4 on the MVME712-10 or MVME712-06 transition module), connect the appropriate cables and configure the port(s) as detailed in step 4. above. After power-up, this(these) port(s) can be reconfigured by programming the MVME166 CD2401 Serial Controller Chip (SCC), or by using the 166Bug **PF** command.

   Note that the MVME166 also contains a parallel port. To use a parallel device, such as a printer, with the MVME166, connect it to the PRINTER/PARALLEL port on an MVME712-10 or MVME712-07 transition module. Refer to the *MVME166/MVME167/MVME187 Single Board Computers Programmer's Reference Guide* for some possible connection diagrams. However, you could also use a module such as the MVME335 for a parallel port connection.

6. Power up the system. 166Bug executes some self-checks and displays the debugger prompt "166-Bug>" (if 166Bug is in Board Mode). However, if the **ENV** command (Appendix A) has put 166Bug in System Mode, the system performs a selftest and tries to autoboot. Refer to the **ENV** and **MENU** commands. They are listed in Table 4-3.

   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.

BOOTBUG

166BBug Implementation

The MVME166 board has a byte-wide EPROM in addition to the Flash memory used to contain the debugger and diagnostics firmware. The EPROM supplied contains the BootBug product (166BBug). Since Flash memory can be electronically erased, the firmware contained in this EPROM is a miniature version of the regular debugger product. It contains enough functionality to enable downloading of object code (by means of the VMEbus, serial port, SCSI bus, or the network) and reprogramming of the Flash memory.

The following table lists the the new commands available in the 166BBug product.

Command	Description
EXEC	Execute User Program
SETUP	Set "System" Parameters

Detailed descriptions of additional subset commands can be found in the *Debugging Package for Motorola 68K CISC CPUs User's Manual*.

There is a jumper on the MVME166 board that controls the operation of the BootBug. If the jumper at J3 pins 7 and 8 is in place (GPI3), then the BootBug (which always executes at reset and powerup) will unconditionally jump to the debugger product contained in the Flash memory. If this jumper is removed, execution will continue in the (diminished functionality) BootBug.

Before using some of the features of the MVME166 BootBug, some parameters may need to be defined. For example, the SCSI ID, the Ethernet address, the clock speed of the board, or possibly the mapping of VMEbus. There is a new command provided for this purpose, the **setup** command. You should run this command and answer the prompts to be sure the board is configured properly before using any SCSI, VME, or Ethernet I/O.

Execute User Program

EXEC *[ADDR]*

The **EXEC** command is used to initiate target code execution. The specified address ("ADDR") is placed in the target Program Counter (PC). Execution will start at the target PC address.

Setup System Parameters

SETUP

Setup allows configuring certain parameters that are necessary for some I/O operations (SCSI, VME, and Ethernet). When you execute this command, the default value is displayed, if any is available, and then you are prompted for input.

The **SETUP** command VME parameters do not stay through a reset. These parameters are not saved to NVRAM. The remaining parameters (MPU Clock Speed, Ethernet Address, Local SCSI Identifier) are saved to NVRAM, but are not checksummed.

```
166-Bug>setup
MPU Clock Speed = "3300"?
Ethernet Address = 000000000000?
Local SCSI Identifier = "07"?
VME Slave Enable [Y/N]                 = N?
VME Slave Starting Address             = 00000000?
VME Slave Ending Address               = 00000000?
VME Slave Address Translation Address  = 00000000?
VME Slave Address Translation Select   = 00000000?
VME Slave Control                      = 0000?
VME Master Enable [Y/N]                = Y?
VME Master Starting Address            = 40000000?
VME Master Ending Address              = 4FFFFFFF?
VME Master Address Translation Address = 00000000?
VME Master Address Translation Select  = 00000000?
VME Master Control                     = 0D?
```

Autoboot

Autoboot is a software routine that is contained in the 166Bug Flash / ROM firmware to provide an independent mechanism for booting an operating system. This autoboot routine automatically scans for controllers and devices in a specified sequence until a valid bootable device containing a boot media is found or the list is exhausted. If a valid bootable device is found, a boot from that device 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.

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

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

Following this message there is a delay to allow you an opportunity to abort the Autoboot process if you wish. Then the actual I/O is begun: the program pointed to within the volume ID of the media specified is loaded into RAM and control passed to it. If, however, during this time you want to gain control without Autoboot, you can press the <BREAK> key or the software ABORT or RESET switches.

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.

Caution **Although streaming tape can be used to autoboot, the same power supply must be connected to the streaming tape drive, controller, and the MVME166. At power-up, the tape controller will position the streaming tape to load point where the volume ID can correctly be read and used.**

If, however, the MVME166 loses power but the controller does not, and the tape happens to be at load point, the sequences of commands required (attach and rewind) cannot be given to the controller and autoboot will not be successful.

ROMboot

On the MVME166, if you want to add other firmware applications, you must note that anytime Flash memory is programmed, the entire contents of Flash will be erased, including the 166Bug product! You should make use of the "block move" command (BM) to copy the debugger from Flash to RAM, combine your own object with the debugger in RAM, and then reprogram Flash from this combined image.

This function is configured / enabled by the Environment (**ENV**) command (refer to Appendix A) and executed at power-up (optionally also at reset) or by the **RB** command assuming there is valid code in the Flash (or optionally elsewhere on the module or VMEbus) to support it. If ROMboot code is installed, a user-written routine is given control (if the routine meets the format requirements). One use of ROMboot might be resetting SYSFAIL* on an unintelligent controller module. The **NORB** command disables the function.

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

a. Power must have just been applied (but the **ENV** command can change this to also respond to any reset).

b. Your routine must be located within the MVME166 ROM memory map (but the **ENV** command can change this to any other portion of the onboard memory, or even offboard VMEbus memory).

c. The ASCII string "BOOT" must be located within the specified memory range.

d. Your routine must pass a checksum test, which ensures that this routine was really intended to receive control at power-up.

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

Network Boot

Network Auto Boot is a software routine contained in the 166Bug EPROMs that provides a mechanism for booting an operating system using a network (local Ethernet interface) as the boot device. The Network Auto Boot routine automatically scans for controllers and devices in a specified sequence until a valid bootable device containing a boot media is found or the list is exhausted. If a valid bootable device is found, a boot from that device 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.)

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

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

Following this message there is a delay to allow you to abort the Auto Boot process if you wish. Then the actual I/O is begun: the program pointed to within the volume ID of the media specified is loaded into RAM and control passed to it. If, however, during this time you want to gain control without Network Boot, you can press the <BREAK> key or the software ABORT or RESET switches.

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 for more details.

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 upon a reset condition. This feature is activated by depressing the RESET and ABORT switches at the same time. This feature instructs the debugger to use the default setup/operation parameters in ROM versus your setup/operation parameters in NVRAM. This feature can be used in the event your setup/operation parameters are corrupted or do not meet a sanity check. Refer to the **ENV** command (Appendix A) for the ROM defaults.

Reset

Pressing and releasing the MVME166 front panel RESET switch initiates a system reset. COLD and WARM reset modes are available. By default, 166Bug is in COLD mode. During COLD reset, a total system initialization takes place, as if the MVME166 had just been powered up. All static variables (including disk device and controller parameters) are restored to their default states. The breakpoint table and offset registers are cleared. The target

registers are invalidated. Input and output character queues are cleared. Onboard devices (timer, serial ports, etc.) are reset, and the *first* two serial ports are reconfigured to their default state.

During WARM reset, the 166Bug variables and tables are preserved, as well as the target state registers and breakpoints.

Reset must be used if the processor ever halts, or if the 166Bug environment is ever lost (vector table is destroyed, stack corrupted, etc.).

Abort

Abort is invoked by pressing and releasing the ABORT switch on the MVME166 front panel. 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.

Pressing and releasing the ABORT switch generates a local board condition which may interrupt the processor if enabled. The target registers, reflecting the machine state at the time the ABORT switch was pressed, are displayed on the screen. Any breakpoints installed in your code are removed and the breakpoint table remains intact. Control is returned to the debugger.

Break

A "Break" is generated by pressing and releasing the BREAK key on the terminal keyboard. Break does not generate an interrupt. The only time break is recognized is when characters are sent or received by the console port. Break removes any breakpoints in your code and keeps the breakpoint table intact. Break also 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.

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.

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 housed in NVRAM (refer to the **CNFG** command in Appendix A). If the check fails, a warning message is displayed. The calculated clock speed is also checked against known clock speeds and tolerances.

Memory Requirements

The program portion of 166Bug is approximately 512KB of code, consisting of download, debugger, and diagnostic packages and contained entirely in Flash. The Flash memory on the MVME166 is mapped starting at location $FF800000.

166Bug 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 166Bug stack and static variable space and the rest is reserved as user space. Whenever the MVME166 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 166Bug initial stack.

Terminal Input/Output Control

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

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

^X	(cancel line)	The cursor is backspaced to the beginning of the line. If the terminal port is configured with the hardcopy or TTY option (refer to PF command), then a carriage return and line feed is issued along with another prompt.
^H	(backspace)	The cursor is moved back one position. The character at the new cursor position is erased. If the hardcopy option is selected, a "/" character is typed along with the deleted character.
<DEL>	(delete or rubout)	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 166Bug 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 166Bug, but you may change them with the **PF** command. In the initialized (default) mode, operation is as follows:

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

Disk I/O Support

166Bug can initiate disk input/output by communicating with intelligent disk controller modules over the VMEbus. Disk support facilities built into 166Bug consist of command-level disk operations, disk I/O system calls (only via one of the TRAP #15 instructions) for use by user programs, and defined data structures for disk parameters.

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 166Bug. Default values for these parameters are assigned at power-up and cold-start reset, but may be altered as described in the section on default parameters, later in this chapter.

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 166Bug 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. 166Bug 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 166Bug Commands

These following 166Bug 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 166Bug 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 166Bug. It is used as a debugging tool.

Disk I/O via 166Bug System Calls

All operations that actually access the disk are done directly or indirectly by 166Bug 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.

To perform a disk operation, 166Bug 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 166Bug Controller and Device Parameters

166Bug 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:

- Use **BO** or **BH**. When you invoke one of these commands, the configuration area of the disk is read and the parameters corresponding to that device are rewritten according to the parameter information contained in the configuration area. This is a temporary change. If a cold-start reset occurs, then the default parameter information is written back into the tables.
- Use the **IOT**. You can use this command to reconfigure the parameter table manually for any controller and/or device that is different from the default. This is also a temporary change and is overwritten if a cold-start reset occurs.
- Obtain the source. You can then change the configuration files and rebuild 166Bug so that it has different defaults. Changes made to the defaults are permanent until changed again.

Disk I/O Error Codes

166Bug 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 (capabilities) and the dependencies of these modules that support the overall network boot function are described in the following paragraphs.

Intel 82596 LAN Coprocessor Ethernet Driver

This driver manages/surrounds the Intel 82596 LAN Coprocessor. Management is in the scope of 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 (next paragraph).

BOOTP Protocol Module

The Bootstrap Protocol (BOOTP) basically allows a diskless client machine to discover its own IP address, the address of a server host, 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 (capabilities) 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

166Bug returns an error code if an attempted network operation is unsuccessful.

Multiprocessor Support

The MVME166 dual-port RAM feature makes the shared RAM available to remote processors as well as to the local processor. This can be done by either of the following two methods. Either method can be enabled / disabled by the **ENV** command as its Remote Start Switch Method (refer to Appendix A).

Multiprocessor Control Register (MPCR) Method

A remote processor can initiate program execution in the local MVME166 dual-port RAM by issuing a remote **GO** command using the Multiprocessor Control Register (MPCR). The MPCR, located at shared RAM location of $800 offset from the base address the debugger loads it at, contains one of two longwords used to control communication between processors. The MPCR contents are organized as follows:

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

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

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

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

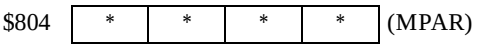
HEX	0	(HEX 00)	--	Wait. Initialization not yet complete.
ASCII	E	(HEX 45)	--	Code pointed to by the MPAR address is executing.
ASCII	P	(HEX 50)	--	Program Flash Memory. The MPAR is set to the address of the Flash memory program control packet.
ASCII	R	(HEX 52)	--	Ready. The firmware monitor is watching for a change.

You can only program Flash memory by the MPCR method. Refer to the .PFLASH system call in the *Debugging Package for Motorola 68K CISC CPUs User's Manual* for a description of the Flash memory program control packet structure.

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

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

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



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

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

If no terminal is connected to the port, the MPCR is still polled to 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.)

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.

GCSR Method

A remote processor can initiate program execution in the local MVME166 dual-port RAM by issuing a remote **GO** command using the VMEchip2 Global Control and Status Registers (GCSR). The remote processor places the MVME166 execution address in general purpose registers 0 and 1 (GPCSR0 and GPCSR1). The remote processor then sets bit 8 (SIG0) of the VMEchip2 LM/SIG register. This causes the MVME166 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 MVME166. The execution address is formed by reading the GCSR general purpose registers in the following manner:

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

The address appears as:

GPCSR0	GPCSR1

Diagnostic Facilities

The 166Bug package includes a complete set of hardware diagnostics intended for testing and troubleshooting of the MVME166. These diagnostics are completely described in the *MVME167Bug Debugging Package User's Manual*. To use the diagnostics, 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 ("`166-Diag>`") appears. 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. The documentation for such diagnostics includes restart information.

USING THE 166Bug DEBUGGER 4

Entering Debugger Command Lines

166Bug is command-driven and performs its various operations in response to user commands entered at the keyboard. When the debugger prompt (166-Bug>) appears on the terminal screen, then the debugger is ready to accept commands.

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

When 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".

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

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

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

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

d. Any required arguments, as specified by command.

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

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

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

Syntactic Variables

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

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

Expression as a Parameter

An expression can be one or more numeric values separated by the arithmetic operators: plus (**+**), minus (**-**), multiplied by (*****), divided by (**/**), logical AND (**&**), shift left (**<<**), or shift right (**>>**).

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

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

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

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

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

Evaluation of an expression is 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 166Bug 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 4-1 summarizes the address formats which are acceptable for address parameters in debugger command lines.

Table 4-1. Debugger Address Parameter Formats

Format	Example	Description
N	140	Absolute address+contents of automatic offset register.
N+Rn	130+R5	Absolute address+contents of the specified offset register (not an assembler-accepted syntax).
(An)	(A1)	Address register indirect. (also post-increment, predecrement)
(d,An) or d(An)	(120,A1) 120(A1)	Address register indirect with displacement (two formats accepted).
(d,An,Xn) or d(An,Xn)	(&120,A1,D2) &120(A1,D2)	Address register indirect with index and displacement (two formats accepted).
([bd,An,Xn],od)	([C,A2,A3],&100)	Memory indirect preindexed.
([bd,An],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:		
([,An],od)	([,A1],4)	
([bd])	([FC1E])	
([bd,,Xn])	([8,,D2])	

NOTES: N — Absolute address(any valid expression).
An — Address register n.
Xn — Index register n (An or Dn).
d — Displacement (any valid expression).
bd — Base displacement (any valid expression).
od — Outer displacement (any valid expression).
n — Register number (0 to 7).
Rn — 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
******  TOTAL ERRORS     0--
******  TOTAL WARNINGS   0--
```

The above program was loaded at address $0001327C.

The disassembled code is shown next:

```
166Bug>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
166Bug>
```

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

```
166Bug>OF R0
R0 =00000000 00000000? 1327C. <CR>
166Bug>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
166Bug>
```

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

Port Numbers

Some 166Bug 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. MVME166 EIA-232-D Debug (Terminal Port 0 or 00) (PORT 1 on the MVME166 J9 connector). Sometimes known as the "console port", it is used for interactive user input/output by default.
2. MVME166 EIA-232-D (Terminal Port 1 or 01) (PORT 2 on the MVME166 J9 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 MVME166 module as "SERIAL PORT 1" and "SERIAL PORT 2", respectively. Physically, they are all part of connector J9.**

Entering and Debugging Programs

There are various ways to enter a user program into system memory for execution. One way is to create the program using the Memory Modify (**MM**) command with the assembler / disassembler option. You enter the program one source line at a time. After each source line is entered, it is assembled and the object code is loaded to memory. Refer to the *Debugging Package for Motorola 68K CISC CPUs User's Manual* for complete details of the 166Bug Assembler / Disassembler.

Another way to enter a program is to download an object file from a host system. The program must be in S-record format (described in the *Debugging Package for Motorola 68K CISC CPUs User's Manual*) and may have been assembled or compiled on the host system. Alternately, the program may have been previously created using the 166Bug **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 MVME166 port 1. (Hardware configuration details are in the section on *Installation and Startup* in Chapter 3.) The file is downloaded from the host to MVME166 memory by the Load (**LO**) command.

Another way is by reading in 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.

Yet another way is via the network, using one of the network disk commands (**NBO**, **NBH**, **NIOP**).

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 166Bug 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. 166Bug uses certain of the MVME166 onboard resources and also offboard system memory to contain temporary variables, exception vectors, etc. If you disturb resources upon which 166Bug 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.

166Bug Vector Table and Workspace

As described in the *Memory Requirements* section in Chapter 3, 166Bug needs 64KB of read / write memory to operate. The 166Bug reserves a 1024-byte area for a user program vector table area and then allocates another 1024-byte area and builds an exception vector table for the debugger itself to use. Next, 166Bug reserves space for static variables and initializes these static variables to predefined default values. After the static variables, 166Bug allocates 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. You should refer to the *Memory Requirements* section in Chapter 3 to determine how to dictate the location of the reserved memory areas. If, for example, your program inadvertently wrote over the static variable area containing the serial communication parameters, these parameters would be lost, resulting in a loss of communication with the system console terminal. If your program corrupts the system stack, then an incorrect value may be loaded into the processor Program Counter (PC), causing a system crash.

Hardware Functions

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

Exception Vectors Used by 166Bug

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 4-2. Exception Vectors Used by 166Bug

Vector Offset	Exception	166Bug Facility
$10	Illegal instruction	Breakpoints (used by **GO**, **GN**, **GT**)
$24	Trace	Trace operations (such as **T**, **TC**, **TT**)
$80-$B8	TRAP #0 - #14	Used internally
$BC	TRAP #15	System calls
$NOTE	Level 7 interrupt	ABORT 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 4-2, 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.

```
166Bug>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
166Bug>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
166Bug>
```

Notice that the value of the target stack pointer register (A7) has not changed even though a trace exception has taken place. Your program may either use the exception vector table provided by 166Bug or it may create a separate exception vector table of its own. The two following sections detail these two methods.

Using 166Bug Target Vector Table

The 166Bug initializes and maintains a vector table area for target programs. A target program is any program started by the bug, either manually with **GO** or **TR** type commands or automatically with the **BO** command. The start address of this target vector table area is the base address of the debugger memory. This address is loaded into the target-state VBR at power up and cold-start reset and can be observed by using the **RD** command to display the target-state registers immediately after power up.

The 166Bug initializes the target vector table with the debugger vectors listed in Table 4-2 and fills the other vector locations with the address of a generalized exception handler (refer to the *166Bug Generalized Exception Handler* section in this chapter). The target program may take over as many vectors as desired by simply writing its own exception vectors into the table. If the vector locations listed in Table 4-2 are overwritten then the accompanying debugger functions are lost.

The 166Bug 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 166Bug vector table into the corresponding vector locations in your program vector table.

The vector for the 166Bug generalized exception handler (described in detail in the *166Bug Generalized Exception Handler* section in this chapter) may be copied from offset $08 (bus error vector) in the target vector table to all locations in your program vector table where a separate exception handler is not used. This provides diagnostic support in the event that your program is stopped by an unexpected exception. The generalized exception handler gives a formatted display of the target registers and identifies the type of the exception.

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

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

It may 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 166Bug target program vector table (which your program saved), yielding the address of the 166Bug exception vector. The program then jumps to the address stored at this vector location, which is the address of the 166Bug exception handler.

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

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

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


166Bug Generalized Exception Handler

The 166Bug has a generalized exception handler which it uses to handle all of the exceptions not listed in Table 4-2. For all these exceptions, the target stack pointer is left pointing to the top of the exception stack frame created. In this way, if an unexpected exception occurs during execution of your code, you are presented with the exception stack frame to help determine the cause of the exception. The following example illustrates this:

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

```
166Bug>RD
PC   =00010000 SR   =2708=TR:OFF_S._7_.N...  VBR  =00000000
USP  =0000DFFC MSP  =0000EFFC ISP* =0000FFFC SFC  =0=F0
DFC  =0=F0     CACR =0=........
D0   =00000001 D1   =00000001 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000002 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =0000FFFC
00010000 203900F0 0000      MOVE.L      ($F00000).L,D0
166Bug>T

Exception: Access Fault (Local Off Board)
PC =FF839154 SR =2704
Format/Vector =7008
SSW =0145 Fault Address =00F00000 Effective Address =0000D4E8
PC   =00010000 SR   =2708=TR:OFF_S._7_.N...  VBR  =00000000
USP  =0000DFFC MSP  =0000EFFC ISP* =0000FFFC SFC  =0=F0
DFC  =0=F0     CACR =0=........
D0   =00000001 D1   =00000001 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000002 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =0000FFC0
00010000 203900F0 0000      MOVE.L      ($F00000).L,D0
166Bug>
```

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

```
166Bug>MD (A7):&30
0000FFC0 2708 0001 0000 7008  0000 FFFC 0105 0005  '.....p.........
0000FFD0 0005 0005 00F0 0000  0000 0A64 0000 FFF4  ...........d....
0000FFE0 00F0 0000 FFFF FFFF  00F0 0000 FFFF FFFF  ................
0000FFF0 2708 0001 A708 0001  0000 0000             '...........
166Bug>
```

Floating Point Support

The floating point unit (FPU) of the MC68040 microprocessor chip is supported in 166Bug. For MVME166Bug, 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 / 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 *M68040 Microprocessor User's Manual*.

The 166Bug Debugger Command Set

The 166Bug debugger commands are summarized in Table 4-3. 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 4-3. 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* [*increment*] [;**B** | **W** | **L**]
CM	Concurrent Mode	**CM** [[*PORT*][*DEL ID-STRING*][*DEL BAUD*] [*DEL PHONE-NUMBER*]] | [;**A**] | [;**H**]
NOCM	No Concurrent Mode	**NOCM**
CNFG	Configure Board Information Block	**CNFG** [;[**I**][**M**]]
CS	Checksum	**CS** *RANGE* [;**B** | **W** | **L**]
DC	Data Conversion	**DC** *EXP* | *ADDR* [;[**B**][**O**][**A**]]
DMA	DMA Block of Memory Move	**DMA** *RANGE DEL ADDR DEL VDIR DEL AM DEL BLK* [;**B** | **W** | **L**]
DS	One Line Disassembler	**DS** *ADDR* [:*COUNT* | *DEL ADDR*]
DU	Dump S-records	**DU** [*PORT*]*DEL RANGE* [*DEL TEXT*][*DEL ADDR*] [*DEL OFFSET*][;**B** | **W** | **L**]
ECHO	Echo String	**ECHO** [*PORT*]*DEL*{*hexadecimal number*} {*'string'*}
ENV	Set Environment to Bug / Operating System	**ENV** [;[**D**]]
GD	Go Direct (Ignore Breakpoints)	**GD** [*ADDR*]

Table 4-3. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
GN	Go to Next Instruction	**GN**
GO	Go Execute User Program	**GO** [*ADDR*]
GT	Go to Temporary Breakpoint	**GT** *ADDR*
HE	Help	**HE** [*COMMAND*]
IOC	I/O Control for Disk	**IOC**
IOI	I/O Inquiry	**IOI** [;[**C**|**L**]]
IOP	I/O Physical (Direct Disk Access)	**IOP**
IOT	I/O "TEACH" for Configuring Disk Controller	**IOT** [;[**A**][**F**][**H**][**T**]]
IRQM	Interrupt Request Mask	**IRQM** [*MASK*]
LO	Load S-records from Host	**LO** [*n*] [*ADDR*] [**;X**|**C**|**T**] [=*text*]
MA	Macro Define/Display	**MA** [*NAME*|**;L**]
NOMA	Macro Delete	**NOMA** [*NAME*]
MAE	Macro Edit	**MAE** *name line#* [*string*]
MAL	Enable Macro Expansion Listing	**MAL**
NOMAL	Disable Macro Expansion Listing	**NOMAL**
MAW	Save Macros	**MAW** [*controller LUN*][*DEL*[*device LUN*][*DEL block #*]]
MAR	Load Macros	**MAR** [*controller LUN*][*DEL*[*device LUN*][*DEL block #*]]
MD	Memory Display	**MD**[**S**] *ADDR*[:*COUNT* | *ADDR*] [; [**B**|**W**|**L**|**S**|**D**|**X**|**P**|**DI**]]
MENU	Menu	**MENU**
MM	Memory Modify	**MM** *ADDR*[;[[**B**|**W**|**L**|**S**|**D**|**X**|**P**][**A**][**N**]]|[**DI**]]
MMD	Memory Map Diagnostic	**MMD** *RANGE DEL increment*[;**B**|**W**|**L**]
MS	Memory Set	**MS** *ADDR* {*Hexadecimal number*} {*'string'*}
MW	Memory Write	**MW** *ADDR DATA* [;**B**|**W**|**L**]
NAB	Automatic Network Boot Operating System	**NAB**
NBH	Network Boot Operating System and Halt	**NBH** [*Controller LUN*][*Device LUN*][*Client IP Address*] [*Server IP Address*][*String*]
NBO	Network Boot Operating System	**NBO** [*Controller LUN*][*Device LUN*][*Client IP Address*] [*Server IP Address*][*String*]
NIOC	Network I/O Control	**NIOC**
NIOP	Network I/O Physical	**NIOP**
NIOT	Network I/O Teach	**NIOT** [;[**H**]|[**A**]]
NPING	Network Ping	**NPING** *Controller-LUN Device-LUN Source-IP Destination-IP* [*N-Packets*]
OF	Offset Registers Display/Modify	**OF** [**R***n*[;**A**]]
PA	Printer Attach	**PA** [*n*]
NOPA	Printer Detach	**NOPA** [*n*]

Table 4-3. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
PF	Port Format	**PF** [*PORT*]
NOPF	Port Detach	**NOPF** [*PORT*]
PFLASH	Program FLASH Memory	**PFLASH** *SSADDR SEADDR DSADDR* [*IEADDR*][**;**[**A** | **R**][**X**]] or **PFLASH** *SSADDR*:*COUNT DSADDR* [*IEADDR*] [**;**[**B** | **W** | **L**] [**A** | **R**] [**X**]]
PS	Put RTC Into Power Save Mode for Storage	**PS**
RB	ROMboot Enable	**RB**[**;V**]
NORB	ROMboot Disable	**NORB**
RD	Register Display	**RD** {[+ | - | =][*DNAME*][*/*]} {[+ | - | =][*REG1*[-*REG2*]][*/*]} [**;E**]
REMOTE	Connect the Remote Modem to CSO	**REMOTE**
RESET	Cold / Warm Reset	**RESET**
RL	Read Loop	**RL** *ADDR*;[**B** | **W** | **L**]
RM	Register Modify	**RM** [*REG*] [;[**S** | **D**]]
RS	Register Set	**RS** ***REG*** [*DEL EXP* | *DEL ADDR*][;[**S** | **D**]]
SD	Switch Directories	**SD**
SET	Set Time and Date	**SET** *mmddyyhhmm* or **SET** *n*;**C**
SYM	Symbol Table Attach	**SYM** [*ADDR*]
NOSYM	Symbol Table Detach	**NOSYM**
SYMS	Symbol Table Display / Search	**SYMS** [*symbol-name*] | [**;S**]
T	Trace	**T** [*COUNT*]
TA	Terminal Attach	**TA** [*port*]
TC	Trace on Change of Control Flow	**TC** [*count*]
TIME	Display Time and Date	**TIME** [;[**C** | **L** | **O**]]
TM	Transparent Mode	**TM** [*n*] [*ESCAPE*]
TT	Trace to Temporary Breakpoint	**TT** ***ADDR***
VE	Verify S-records Against Memory	**VE** [*n*] [*ADDR*] [**;**[**X**][**C**]] =*text*]
VER	Display Revision / Version	**VER** [**; E**]
WL	Write Loop	**WL** *ADDR*:*DATA*;[**B** | **W** | **L**]

CONFIGURE AND ENVIRONMENT COMMANDS

A

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 *MVME166 Single Board Computer User's Manual* for the actual location. The information block contains various elements detailing specific operation parameters of the hardware. The *MVME166 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.

```
166-Bug>cnfg
Board (PWA) Serial Number = "000000061050"
Board Identifier          = "MVME166-11      "
Artwork (PWA) Identifier  = "01-W3834B01B    "
MPU Clock Speed           = "3300"
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       = "        "
166-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. Some of these parameters are set up by the factory, and correct board operation relies upon these parameters.

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

Set Environment to Bug/Operating System

ENV [;[**D**]]

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

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

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

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

The 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 (hard disk only)
Auto Boot Enable [Y / N]	N	Auto Boot function is disabled.
Auto Boot at power-up only [Y / N]	Y	Auto Boot is attempted at power up reset only.
Auto Boot Controller LUN	00	LUN of a disk / tape controller module currently supported by the Bug. Default is $0.
Auto Boot Device LUN	00	LUN of a disk / tape device currently supported by the Bug. Default is $0.
Auto Boot Abort Delay	15	This is the time in seconds that the Auto Boot sequence will delay before starting the boot. The purpose for the delay is to allow you the option of stopping the boot by use of the Break key. The time value is from 0 through 255 seconds.

Table A-1. ENV Command Parameters (Continued)

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

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Network Autoboot Configuration Parameters Pointer (NVRAM)	00000000	This is the address where the network interface configuration parameters are to be saved / retained in NVRAM; these parameters are the necessary parameters to perform an unattended network boot.
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-166 environment, each MVME166 board could be set to start its work page at a unique address to allow multiple debuggers to operate simultaneously.
Memory Search Ending Address	02000000	Top limit of the Bug's search for a work page. If a contiguous block of memory, 64KB in size, is not found in the range specified by Memory Search Starting Address and Memory Search Ending Address parameters, then the bug will place its work page in the onboard static RAM on the MVME166. 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.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
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	Default address is $FFFFCE0F. This is the MVME166 GCSR GPCSR0 as accessed through VMEbus A16 space and assumes the MVME166 GRPAD (group address) and BDAD (board address within group) switches are set to "on". This byte-wide value is initialized to $FF by MVME166 hardware after a System or Power-on Reset. In a multi-166 environment, where the work pages of several Bugs are to reside in the memory of the primary (first) MVME166, the non-primary CPUs will wait for the data at the Memory Search Delay Address to be set to $00, $01, or $02 (refer to the *Memory Requirements* section in Chapter 3 for the definition of these values) before attempting to locate their work page in the memory of the primary CPU.
Memory Size Enable [Y / N]	Y	Memory will be sized for Self Test diagnostics.
Memory Size Starting Address	00000000	Default Starting Address is $0.
Memory Size Ending Address	02000000	Default Ending Address is the calculated size of local memory.
Base Address of Local Memory	00000000	Beginning address of Local Memory. It must be a multiple of the Local Memory board size, starting with 0. The Bug will set up hardware address decoders so that Local Memory resides as one contiguous block at this address. Default is $0.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Size of Local Memory Board #0 Size of Local Memory Board #1	02000000 00000000	You are prompted twice, once for each possible MVME166 memory board. Default is the calculated size of the memory board.
Slave address decoders setup. The slave address decoders are use to allow another VMEbus master to access a local resource of the MVME166. There are two slave address decoders set. 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. Default is the base of local memory, $0.
Slave Ending Address #1	01FFFFFF	Ending address of the local resource that is accessible by the VMEbus. Default is the end of calculated memory.
Slave Address Translation Address #1	00000000	This register will allow the VMEbus address and the local address to be different. The value in this register is the base address of local resource that is associated with the starting and ending address selection from the previous questions. Default is 0.
Slave Address Translation Select #1	00000000	This register defines which bits of the address are significant. A logical one "1" indicates significant address bits, logical zero "0" is non-significant. Default is 0.
Slave Control #1	03FF	Defines the access restriction for the address space defined with this slave address decoder. Default is $03FF.
Slave Enable #2 [Y / N]	Y	Yes, Setup and enable the Slave Address Decoder #2.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Slave Starting Address #2	FFE00000	Base address of the local resource that is accessible by the VMEbus. Default is the base address of static RAM, $FFE00000.
Slave Ending Address #2	FFE1FFFF	Ending address of the local resource that is accessible by the VMEbus. Default is the end of static RAM, $FFE1FFFF.
Slave Address Translation Address #2	00000000	Works the same as Slave Address Translation Address #1. Default is 0.
Slave Address Translation Select #2	00000000	Works the same as Slave Address Translation Select #1. Default is 0.
Slave Control #2	01EF	Defines the access restriction for the address space defined with this slave address decoder. Default is $01EF.
Master Enable #1 [Y / N]	Y	Yes, 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, unless memory is less than 16MB, then this register will always be set to 01000000.
Master Ending Address #1	EFFFFFFF	Ending address of the VMEbus resource that is accessible from the local bus. Default is the end of calculated memory.
Master Control #1	0D	Defines the access characteristics for the address space defined with this master address decoder. Default is $0D.
Master Enable #2 [Y / N]	N	Do not setup and enable the Master Address Decoder #2.
Master Starting Address #2	00000000	Base address of the VMEbus resource that is accessible from the local bus. Default is $00000000.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Master Ending Address #2	00000000	Ending address of the VMEbus resource that is accessible from the local bus. Default is $00000000.
Master Control #2	00	Defines the access characteristics for the address space defined with this master address decoder. Default is $00.
Master Enable #3 [Y / N]	N	Yes, setup and enable the Master Address Decoder #3. This is the default if the board contains less than 16MB of calculated RAM. Do not set up and enable the Master Address Decoder #3. This is the default for boards containing at least 16MB of calculated RAM.
Master Starting Address #3	00000000	Base address of the VMEbus resource that is accessible from the local bus. If enabled, the value is calculated as one less than the calculated size of memory. If not enabled, the default is $00000000.
Master Ending Address #3	00000000	Ending address of the VMEbus resource that is accessible from the local bus. If enabled, the default is $00FFFFFF, otherwise $00000000.
Master Control #3	00	Defines the access characteristics for the address space defined with this master address decoder. If enabled, the default is $3D, otherwise $00.
Master Enable #4 [Y / N]	N	Do not set up and enable the Master Address Decoder #4.
Master Starting Address #4	00000000	Base address of the VMEbus resource that is accessible from the local bus. Default is $0.
Master Ending Address #4	00000000	Ending address of the VMEbus resource that is accessible from the local bus. Default is $0.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Master Address Translation Address #4	00000000	This register will allow the VMEbus address and the local address to be different. The value in this register is the base address of VMEbus resource that is associated with the starting and ending address selections from the previous questions. Default is 0.
Master Address Translation Select #4	00000000	This register defines which bits of the address are significant. A logical one "1" indicates significant address bits, logical zero "0" is non-significant. Default is 0.
Master Control #4	00	Defines the access characteristics for the address space defined with this master address decoder. Default is $00.
Short I/O (VMEbus A16) Enable [Y/N]	Y	Yes, Enable the Short I/O Address Decoder.
Short I/O (VMEbus A16) Control	01	Defines the access characteristics for the address space defined with the Short I/O address decoder. Default is $01.
F-Page (VMEbus A24) Enable [Y/N]	Y	Yes, Enable the F-Page Address Decoder.
F-Page (VMEbus A24) Control	02	Defines the access characteristics for the address space defined with the F-Page address decoder. Default is $02.
ROM Speed Bank A Code ROM Speed Bank B Code	04 04	Used to set up the ROM speed. Default $04 = 145 ns.
Static RAM Speed Code	00	Used to set up the SRAM speed. Default $00 = 115 ns.
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.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
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 systems controller. Default $01 = 64 µs.
Local Bus Time Out Code	00	This controls the local bus timeout. Default $00 = 8 µs.
VMEbus Access Time Out Code	02	This controls the local bus to VMEbus access timeout. Default $02 = 32 ms.
The special 166Bug parameters that can be configured using **ENV** are as follows.		
VSBC2 Installed [Y/N]	Y	This is set depending on the existence of a VSBchip2 on the board. It can be overridden with this **ENV** option.
VSBC2 Interrupt Vector Base	0E	Vector passed back to the VSB master during the Status/ID transfer phase of an interrupt-acknowledge cycle, if the VSBchip2 wins interrupt arbitration. Default = 0E.
VSBC2 Local Interrupt Vector Base	00	The value here is part of the interrupt vector supplied on the local bus during an interrupt acknowledge cycle. The lower 4 bits are reserved (and are read-only, as zeroes). Bits 4 through 7 are programmable. Default = 00.
VSBC2 Slave Starting Address #1	00000000	Beginning address of an address range for the first VSB to local bus map decoder. Only the upper 16 bits of this field are significant (or used). Default = 00000000.
VSBC2 Slave Ending Address #1	00000000	Ending address of an address range for the first VSB to local bus map decoder. Only the upper 16 bits of this field are significant (or used). Default = 00000000.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
VSBC2 Slave Address Offset #1	00000000	Address offset for the first VSB to local bus map decoder. The upper 16 bits of this field will be added to the upper 16 bits of the VSB address received. This sum is then the address driven onto the local bus address lines. Default = 00000000.
VSBC2 Slave Attributes #1	0400	The bits in this register control various aspects of how the first VSB to local bus map decoder will operate. Consult the VSBchip2 register definition in the MVME166 programmer's reference guide for an explanation of each bit. Default = 0400.
VSBC2 Slave Starting Address #2	00000000	Beginning address of an address range for the second VSB to local bus map decoder. Only the upper 16 bits of this field are significant (or used). Default = 00000000.
VSBC2 Slave Ending Address #2	00000000	Ending address of an address range for the second VSB to local bus map decoder. Only the upper 16 bits of this field are significant (or used). Default = 00000000.
VSBC2 Slave Address Offset #2	00000000	Address offset for the second VSB to local bus map decoder. The upper 16 bits of this field will be added to the upper 16 bits of the VSB address received. This sum is then the address driven onto the local bus address lines. Default = 00000000.
VSBC2 Slave Attributes #2	0400	The bits in this register control various aspects of how the second VSB to local bus map decoder will operate. Consult the VSBchip2 register definition in the MVME166 programmer's reference guide for an explanation of each bit. Default = 0400.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
VSBC2 Requester Control	01000000	The bits in this register control aspects of the local board's access of the VSB. Control and status bits are available to request control of the bus (and determine when control has been obtained), determine whether the requester uses "fairness" mode in arbitrating for the bus, whether to release the bus when done, and configure the VSB requester arbitration ID value. Definition of the register contents can be found in the MVME166 programmer's reference guide. Default = 01000000.
VSBC2 Timer Control Register	1000	The bits in this register control various timers and their timeout periods, for VSB accesses. Definition of the register contents can be found in the MVME166 programmer's reference guide. Default = 1000.
VSBC2 Master Starting Address #1	00000000	Beginning address of an address range for the local bus to VSB map decoder #1. Only the upper 16 bits of this field are significant (or used). Default = 00000000.
VSBC2 Master Ending Address #1	00000000	Ending address of an address range for the local bus to VSB map decoder #1. Only the upper 16 bits of this field are significant (or used). Default = 00000000.
VSBC2 Master Address Offset #1	00000000	Address offset for the local bus to VSB map decoder #1. The upper 16 bits of this field will be added to the upper 16 bits of the local bus address received. This sum is then the address driven onto the VSB address lines. Default = 00000000.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
VSBC2 Master Attributes #1	0030	The bits in this register control various aspects of how the local bus to VSB map decoder #1 will operate. Consult the VSBchip2 register definition in the MVME166 programmer's reference guide for a detailed explanation of each bit. Default = 0030.
VSBC2 Master Starting Address #2	00000000	Beginning address of an address range for the local bus to VSB map decoder #2. Only the upper 16 bits of this field are significant (or used). Default = 00000000.
VSBC2 Master Ending Address #2	00000000	Ending address of an address range for the local bus to VSB map decoder #2. Only the upper 16 bits of this field are significant (or used). Default = 00000000.
VSBC2 Master Address Offset #2	00000000	Address offset for the local bus to VSB map decoder #2. The upper 16 bits of this field will be added to the upper 16 bits of the local bus address received. This sum is then the address driven onto the VSB address lines. Default = 00000000.
VSBC2 Master Attributes #2	0030	The bits in this register control various aspects of how the local bus to VSB map decoder #2 will operate. Consult the VSBchip2 register definition in the MVME166 programmer's reference guide for a detailed explanation of each bit. Default = 0030.
VSBC2 Master Starting Address #3	00000000	Beginning address of an address range for the local bus to VSB map decoder #3. Only the upper 16 bits of this field are significant (or used). Default = 00000000.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
VSBC2 Master Ending Address #3	00000000	Ending address of an address range for the local bus to VSB map decoder #3. Only the upper 16 bits of this field are significant (or used). Default = 00000000.
VSBC2 Master Address Offset #3	00000000	Address offset for the local bus to VSB map decoder #3. The upper 16 bits of this field will be added to the upper 16 bits of the local bus address received. This sum is then the address driven onto the VSB address lines. Default = 00000000.
VSBC2 Master Attributes #3	0030	The bits in this register control various aspects of how the local bus to VSB map decoder #3 will operate. Consult the VSBchip2 register definition in the MVME166 programmer's reference guide for a detailed explanation of each bit. Default = 0030.
VSBC2 Master Starting Address #4	00000000	Beginning address of an address range for the local bus to VSB map decoder #4. Only the upper 16 bits of this field are significant (or used). Default = 00000000.
VSBC2 Master Ending Address #4	00000000	Ending address of an address range for the local bus to VSB map decoder #4. Only the upper 16 bits of this field are significant (or used). Default = 00000000.
VSBC2 Master Address Offset #4	00000000	Address offset for the local bus to VSB map decoder #4. The upper 16 bits of this field will be added to the upper 16 bits of the local bus address received. This sum is then the address driven onto the VSB address lines. Default = 00000000.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
VSBC2 Master Attributes #4	0030	The bits in this register control various aspects of how the local bus to VSB map decoder #4 will operate. Consult the VSBchip2 register definition in the MVME166 programmer's reference guide for a detailed explanation of each bit. Default = 0030.

DISK/TAPE CONTROLLER DATA

B

Disk/Tape Controller Modules Supported

The following VMEbus disk / tape controller modules are supported by the 166Bug. 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 (SBC)	$00 (NOTE 1)	--	--	--
MVME320 - Winchester / Floppy Controller	$11 (NOTE 2)	$FFFFB000	$12 (NOTE 2)	$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

NOTES: (1) If the SBC (e.g., an MVME166) SCSI port is used, then the SBC module has CLUN 0.

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

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	SCSI Common Command Set
		10	(CCS), which may be any of these:
3	$FFFFA700	20	
		30	- Fixed direct access
		40	- Removable flexible direct access
		50	(TEAC style)
		60	- CD-ROM
			- Sequential access
		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, MVME166, 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
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 DATA C

Network Controller Modules Supported

The following VMEbus network controller modules are supported by the MVME166Bug. The default address for each type and position is showed to indicate where the controller must reside to be supported by the MVME166Bug. The controllers are accessed via the specified CLUN and DLUNs listed here. The CLUN and DLUNs are used in conjunction with the debugger commands **NBH**, **NBO**, **NIOP**, **NIOC**, **NIOT**, **NPING**, and **NAB**, and also with the debugger system calls .NETRD, .NETWR, .NETFOPN, .NETFRD, .NETCFIG, and .NETCTRL.

Controller Type	CLUN	DLUN	Address	Interface Type
MVME166	$00	$00	$FFF46000	Ethernet
MVME376	$02	$00	$FFFF1200	Ethernet
MVME376	$03	$00	$FFFF1400	Ethernet
MVME376	$04	$00	$FFFF1600	Ethernet
MVME376	$05	$00	$FFFF5400	Ethernet
MVME376	$06	$00	$FFFF5600	Ethernet
MVME376	$07	$00	$FFFFA400	Ethernet
MVME374	$10	$00	$FF000000	Ethernet
MVME374	$11	$00	$FF100000	Ethernet
MVME374	$12	$00	$FF200000	Ethernet
MVME374	$13	$00	$FF300000	Ethernet
MVME374	$14	$00	$FF400000	Ethernet
MVME374	$15	$00	$FF500000	Ethernet

C

Index

When using this index, keep in mind that a page number indicates only where referenced material begins. It may extend to the page or pages following the page referenced.

D

E

F

G

H

I

J

L

M

N

O

P

INDEX

Q

R

S

T

U

V

W

X