MVME162

Embedded Controller

Installation Guide

(MVME162IG/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 Rights Legend

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

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

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

Preface

This manual provides a general board level hardware description, hardware preparation and installation instructions, debugger general information, and using the debugger in the MVME162 Embedded Controller. The information contained in this manual applies to the following MVME162 models:

MVME162-001	MVME162-010	MVME162-020	MVME162-030	MVME162-040
MVME162-002	MVME162-011	MVME162-021	MVME162-031	MVME162-041
MVME162-003	MVME162-012	MVME162-022	MVME162-032	MVME162-042
	MVME162-013	MVME162-023	MVME162-033	MVME162-043
	MVME162-014	MVME162-026		

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

August 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

BOARD LEVEL HARDWARE DESCRIPTION 1

Introduction

This chapter describes the board level hardware features of the MVME162 Embedded Controller. 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 MVME162 that reside in ASICs are covered in the *MVME162 Embedded Controller Programmer's Reference Guide.*

Overview

The MVME162 is based on the MC68040 or MC68LC040 microprocessor. Various versions of the MVME162 have 1 MB, 4 MB, or 8 MB of parity-protected DRAM, 8 KB of SRAM (with battery backup), time of day clock (with battery backup), Ethernet transceiver interface, two serial ports with EIA-232-D or EIA-530 interface, six tick timers, watchdog timer, a PROM socket, 1 MB Flash memory (one or four Flash devices), four IndustryPack (IP) interfaces, SCSI bus interface with DMA, VMEbus controller, and 512 KB of SRAM with battery backup.

The I/O on the MVME162 is connected to the VMEbus P2 connector. The main board is connected through a P2 transition board and cables to the transition boards. The MVME162 supports the transition boards MVME712-12, MVME712-13, MVME712M, MVME712A, MVME712AM, and MVME712B (referred to in this manual as MVME712X, unless separately specified). The MVME712X transition boards provide configuration headers and provide industry standard connectors for the I/O devices.

The I/O connection for the serial ports on the MVME162 is also provided by two DB-25 front panel I/O connectors. The MVME712 series transition boards were designed to support the MVME167 boards, but can be used on the MVME162 by following some special precautions. (Refer to the section on the Serial Communications Interface, later in this chapter, for more information.) 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 MCchip ASIC provides four tick timers, the interface to the LAN chip, SCSI chip, serial port chip, BBRAM, and the programmable interface for the parity-protected DRAM and/or SRAM mezzanine board.

The IndustryPack Interface Controller (IPIC) ASIC provides control and status information for up to four single size IndustryPacks (IPs) or up to two double size IPs that can be plugged into the MVME162 main module.

Related Documentation

The MVME162 does not ship with all of the documentation that is available for the product. The MVME162 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 MVME162IG/D for the MVME162.

The following publications are applicable to the MVME162 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
MVME162 Embedded Controller User's Manual	MVME162
MVME162 Embedded Controller Support Information	SIMVME162
MVME162Bug Debugging Package User's Manual	MVME162BUG
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
MVME162 Embedded Controller Programmer's Reference Guide	MVME162PG
MVME712M Transition Module and P2 Adapter Board User's Manual	MVME712M
MVME712-12, MVME712-13, MVME712A, MVME712AM, and MVME712B Transition Modules and LCP2 Adapter Board User's Manual	MVME712A
M68040 Microprocessors User's Manual	M68040UM

Notes

The SIMVME162 manual contains the connector interconnect signal information, parts lists, and schematics for the MVME162.

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-MVME162SET for use with the MVME162.

MVME162 / D
MVME162BUG / D
68KBUG / D
SBCSCSI / D
MVME162PG / D
SIMVME162 / D

To further assist your development effort, Motorola has collected user's manuals for each of the peripheral controllers used on the MVME162 and other boards from the suppliers. This bundle includes manuals and data sheets, including the following:

68-1X7DS for use with the MVME162 and 167.

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 available as *Microprocessor system bus for 1 to 4 byte data, IEC 821 BUS*, Bureau Central de la Commission Electrotechnique Internationale; 3, rue de Varembé, Geneva, Switzerland.

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

IndustryPack Logic Interface Specification, Revision 1.0; GreenSpring Computers, Inc., 1204 O'Brien Drive, Menlo Park, CA 94025.

Z85230 Serial Communications Controller data sheet; Zilog, Inc., 210 Hacienda Ave., Campbell, California 95008-6609.

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.

28F008SA Flash Memory Data Sheet, order number 2904351; Intel Literature Sales, P.O. Box 7641, Mt. Prospect, IL 60056-7641.

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
- ❏ IndustryPack Specification, GreenSpring

Features

- ❏ 25MHz 32-bit MC68040 or MC68LC040 Microprocessor
- ❏ 1 MB, 4 MB, or 8 MB of shared DRAM with parity protection
- ❏ 512 KB of SRAM with battery backup
- ❏ One JEDEC standard 32-pin PLCC EPROM socket (EPROMs may be shipped separately from the MVME162)
- ❏ One Intel 28F008SA 1M x 8 Flash memory device or four Intel 28F020 256K x 8 Flash memory devices (1 MB Flash memory total)
- ❏ 8K by 8 Non-Volatile RAM and time of day clock with battery backup
- ❏ Four 32-bit Tick Timers (in the MCchip ASIC) for periodic interrupts
- ❏ Two 32-bit Tick Timers (in the VMEchip2 ASIC) for periodic interrupts
- ❏ Watchdog timer
- ❏ Eight software interrupts (for MVME162 versions that have the VMEchip2)
- ❏ I/O
  - – Two serial ports (one EIA-232-D DCE; one EIA-232-D or EIA-530 DCE/DTE)
  - – Serial port controller (Zilog Z85230)
  - – Optional Small Computer Systems Interface (SCSI) bus interface with 32-bit local bus burst Direct Memory Access (DMA) (NCR 53C710 controller)
  - – Optional LAN Ethernet transceiver interface with 32-bit local bus DMA (Intel 82596CA controller)
  - – Four MVIP IndustryPack interfaces
- ❏ VMEbus interface
  - – VMEbus system controller functions

  - VMEbus interface to local bus (A24 / A32, D8 / D16 / D32 (D8 / D16 / D32 / D64 BLT) (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 / D64 BLT)

- ❑ Switches and Light-Emitting Diodes (LEDs)
  - Two pushbutton switches (ABORT and RESET)
  - Eight LEDs (FAIL, STAT, RUN, SCON, LAN, FUSE, SCSI, and VME)

Specifications

General specifications for the MVME162 are listed in Table 1-1.

Table 1-1. MVME162 Specifications

Characteristics	Specifications
Power requirements (with PROM; without IPs)	+5V (± 5%), 3.5 A typical, 4.5 A max. +12 Vdc (± 5%), 100 mA (max.) -12 Vdc (± 5%), 100 mA (max.)
Operating temperature	0° to 70° C exit air with forced air cooling (see NOTE)
Storage temperature	-40° to +85° C
Relative humidity	5% to 90% (noncondensing)
Physical dimensions	Double-high VMEboard
PC board with mezzanine module only	
Height	9.187 inches (233.35 mm)
Depth	6.299 inches (160.00 mm)
Thickness	0.662 inch (16.77 mm)
PC board with connectors and front panel	
Height	10.309 inches (261.85 mm)
Depth	7.4 inches (188 mm)
Thickness	0.80 inch (20.32 mm)

NOTE: Refer to the following section on “Special Considerations for Elevated Temperature Operation,” and to “Cooling Requirements” in the *MVME162 Embedded Controller User’s Manual*.

Special Considerations for Elevated Temperature Operation

The following information is for the user who has an application for the MVME162 which will subject it to high temperature.

The MVME162 uses commercial grade devices. Therefore, it can operate in an environment with ambient air temperature from 0° C to 70° C. There are many factors that affect the ambient temperature seen by components on the MVME162: inlet air temperature; air flow characteristics; number, types, and locations of IndustryPack (IP) modules; power dissipation of adjacent boards in the system; etc.

A temperature profile was performed for the MVME162-23 in an MVME945 12-slot VME chassis. This board was loaded with one GreenSpring IP-Dual P/T module (position a) and three GreenSpring IP-488 modules (positions b, c, and d). One twenty-five watt load board was installed adjacent to each side of the board under test. The exit air velocity was approximately 200 LFM between the MVME162 and the IP-Dual P/T module. Under these circumstances, a 10° C rise between the inlet and exit air was observed. At 70° C exit air temperature (60° C inlet air), the junction temperatures of devices on the MVME162 were calculated (from the measured case temperatures) and do not exceed 100° C.

Caution **For elevated temperature operation, the user must perform similar measurements and calculations to determine what operating margin exists for any specific environment.**

The following are some steps that the user can take to help make elevated temperature operation possible:

1. Position the MVME162 board in the chassis for maximum airflow over the component side of the board.
2. Avoid placing boards with high power dissipation adjacent to the MVME162.
3. Use low power IP modules only. The preferred locations for IP modules are position a (J2 and J3) and position d (J18 and J19).

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 MVME162 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 MVME162 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 MVME162.

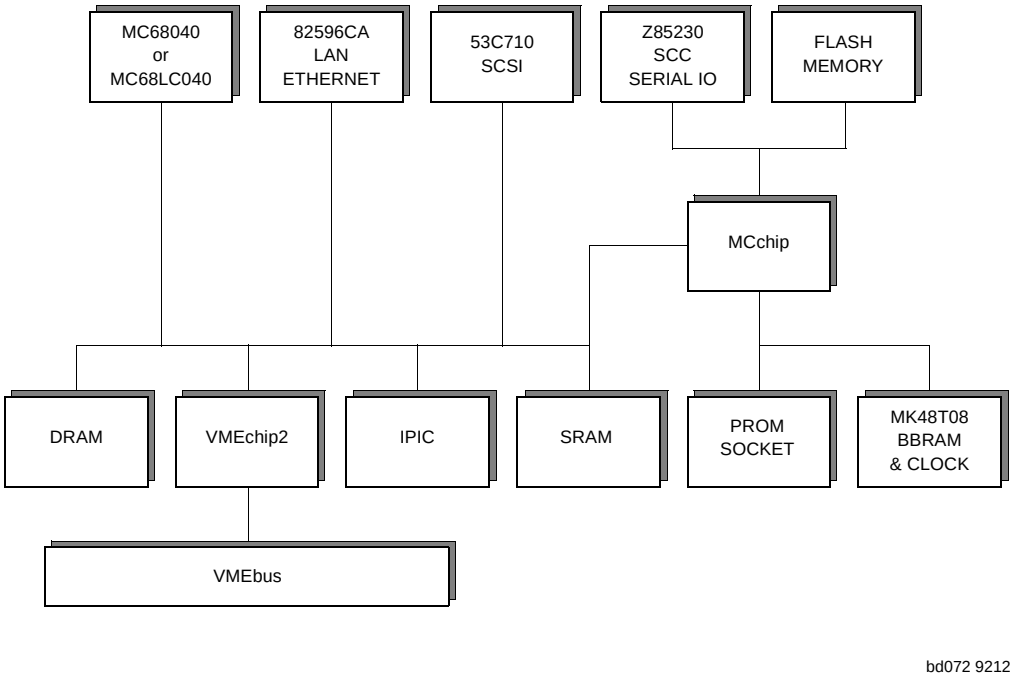


Figure 1-1. MVME162 Block Diagram

Functional Description

This section contains a functional description of the major blocks on the MVME162 Embedded Controller.

Front Panel Switches and Indicators

There are switches and LEDs on the front panel of the MVME162. 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 eight LEDs on the MVME162 front panel: FAIL, STAT, RUN, SCON, LAN, FUSE (LAN power), SCSI, and VME.

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

The MC68040 status lines are decoded, on the MVME162, 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 is the VMEbus system controller.

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

The MVME162 supplies +12Vdc power to the Ethernet transceiver interface through a fuse. The green FUSE (LAN power) LED (part of DS3) lights when power is available to the transceiver interface.

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

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

Data Bus Structure

The local data bus on the MVME162 is a 32-bit synchronous bus that is based on the MC68040 bus, and which 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, 53C710 SCSI, VMEbus, and MPU. Generally speaking, any master can access any slave; however, not all combinations pass the common sense test. Refer to the *MVME162 Embedded Controller 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 or MC68LC040 MPU

The MC68040 or MC68LC040 processor is used on the MVME162. The MC68040 has on-chip instruction and data caches and a floating point processor. The major difference between the two processors is that the MC68040 has a floating point coprocessor. Refer to the *M68040 Microprocessor User's Manual* for more information.

MC68xx040 Cache

The MVME162 local bus masters (VMEchip2, MC68xx040, 53C710 SCSI controller, and 82596CA Ethernet controller) have programmable control of the snoop/caching mode. The MVME162 local bus slaves which support MC68xx040 bus snooping are defined in the Local Bus Memory Map table later in this chapter.

No-VMEbus-Interface Option

The MVME162 can be operated as an embedded controller without the VMEbus interface. To support this feature, certain logic in the VMEchip2 has been duplicated in the MCchip. This logic is inhibited in the MCchip if the VMEchip2 is present. The enables for these functions are controlled by software and MCchip hardware initialization.

Contact your local Motorola sales office for ordering information.

Memory Options

The following memory options are used on the different versions of MVME162 boards.

DRAM Options

The MVME162 implementation includes a 1 MB, 4 MB, or 8 MB DRAM option. The DRAM architecture is non-interleaved for 1 MB and 8 MB; while the 4 MB architecture is interleaved. Parity protection can be enabled with interrupts or bus exception when a parity error is detected. DRAM performance is specified in the section on the DRAM Memory Controller in the MCchip Programming Model in the *MVME162 Embedded Controller Programmer's Reference Guide.*

SRAM Options

The MVME162 implementation includes a 512 KB SRAM option. SRAM architecture is single non-interleaved. SRAM performance is specified in the section on the SRAM Memory Controller in the MCchip Programming Model in the *MVME162 Embedded Controller Programmer's Reference Guide.* A battery supplies VCC to the SRAMs when main power is removed. The worst case elapsed time for battery protection is 200 days.

The SRAM arrays are not parity protected.

The MVME162 SRAM 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 MVME162 provides jumpers (on J20) that allow either power source of the DS1210S to be connected to the VMEbus +5 V STDBY pin or to 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. Refer to Chapter 2 for the jumper configurations.

Caution **For proper operation of the SRAM, some jumper combination must be installed on the Backup Power Source Select Header (J20). If one of the jumpers is used to select the battery, the battery must be installed on the MVME162. The SRAM may malfunction if inputs to the DS1210S are left unconnected.**

The SRAM is controlled by the MCchip, and the access time is programmable. Refer to the MCchip description in the *MVME162 Embedded Controller Programmer's Reference Guide* for more detail.

About the Battery

The power source for the onboard SRAM is a RAYOVAC FB1225 battery with two BR1225 type lithium cells which 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 MVME162 should provide at least two years of backup time with the board powered off and with an ambient temperature of 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 MVME162 is shipped with the batteries disconnected (i.e., with VMEbus +5V standby voltage selected as both primary and secondary power source). If you intend to use the battery as a power source, whether primary or secondary, it is necessary to reconfigure the jumpers on J20 before installing the module. Refer to *SRAM Backup Power Source Select Header J20* in Chapter 2 for available jumper configurations.

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.

WARNING

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, possible 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.

EPROM and Flash

The MVME162 implementation includes 1 MB of Flash memory (an 8-Mbit Flash device organized as a 1M X 8, or four 2-Mbit Flash devices organized as 256Kbit x 8). For information on programming Flash, refer to the Intel documents listed in Related Documentation in this chapter. The EPROM location is a standard JEDEC 32-pin PLCC capable of 4 Mbit densities organized as a 512 KB X 8 device. Depending on a jumper setting (GPIO3, pins 9-10 on J22), the MC68xx040 reset code can be fetched from either the Flash (GPIO3 installed) or EPROM (GPIO3 removed).

Battery Backed Up RAM and Clock

The MK48T08 RAM and clock chip is used on the MVME162. This chip provides a time of day clock, oscillator, crystal, power failure 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- day, 29-day (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 MCchip supports 8-bit, 16-bit, and 32-bit accesses to the MK48T08. Refer to the MCchip description in the *MVME162 Embedded Controller Programmer's Reference Guide* and to the MK48T08 data sheet for detailed programming and battery life information.

VMEbus Interface and VMEchip2

The local bus to VMEbus interface and the VMEbus to local bus interface are provided by the optional VMEchip2. The VMEchip2 can also provide the VMEbus system controller functions. Refer to the VMEchip2 description in the *MVME162 Embedded Controller Programmer's Reference Guide* for detailed programming information.

Note that the ABORT switch logic in the VMEchip2 is not used. The GPI inputs to the VMEchip2 which are located at $FFF40088 bits 7-0 are not used. The ABORT switch interrupt is integrated into the MCchip ASIC at location $FFF42043. The GPI inputs are integrated into the MCchip ASIC at location $FFF4202C bits 23-16.

I/O Interfaces

The MVME162 provides onboard I/O for many system applications. The I/O functions include serial ports, IndustryPack (IP) interfaces, optional LAN Ethernet transceiver interface, and optional SCSI mass storage interface.

Serial Communications Interface

The MVME162 uses a Zilog Z85230 serial communications controller to implement the two serial communications interfaces. Each interface supports CTS, DCD, RTS, and DTR control signals; as well as the TxD and RxD transmit/receive data signals, and TxC/RxC synchronous clock signals.

The Z85230 supports synchronous (SDLC/HDLC) and asynchronous protocols. The MVME162 hardware supports asynchronous serial baud rates of 110b/s to 38.4Kb/s.

The Z85230 supplies an interrupt vector during interrupt acknowledge cycles. The vector is modified based upon the interrupt source within the Z85230. Interrupt request levels are programmed via the MCchip. Refer to the Z85230 data sheet listed in this chapter, and to the MCchip Programming Model in the *MVME162 Embedded Controller Programmer's Reference Guide*, for information.

MVME162 Serial Port 1

The A port of the Z85230 is interfaced as DCE (data circuit-terminating equipment) with the EIA-232-D interface and is routed to:

- ❏ The DB-25 connector marked SERIAL PORT 1/CONSOLE on the front panel of the **MVME162**. SERIAL PORT 1/CONSOLE is an EIA-232-D DCE port.

  NOTE: This port can be connected to the TX and RX clocks which may be present on the DB-25 connector. These connections are made via jumper header J11 on the MVME162 board. (The TxC and RxC clock lines are not available on the MVME712X transition modules.)

- ❏ One of the following output connectors on the MVME712X transition module:

  MVME712M: The DB-25 connector marked SERIAL PORT 2 on the front panel. SERIAL PORT 2 can be configured as an EIA-232-D DTE or DCE port, via jumper headers J16 and J17.

  MVME712A or MVME712-12: The DB-9 connector marked SERIAL PORT 2 on the front panel. SERIAL PORT 2 is hardwired as an EIA-232-D DTE port.

  MVME712AM or MVME712-13: The DB-9 connector marked SERIAL PORT 2 ***OR*** the RJ-11 jack on the front panel. SERIAL PORT 2 is hardwired as EIA-232-D DTE; the RJ-11 jack utilizes the built-in modem. Setting the jumper headers J16 and J17 on the MVME712AM/-13 configures the output as EIA-232-D DTE at SERIAL PORT 2 or as a modem at the RJ-11 jack.

MVME162 Serial Port 2

The configuration of the B port of the Z85230 is determined via a Serial Interface Module (SIM) which is installed at connector J10 on the MVME162 board. There are four SIMs available:

SIM05 -- DTE with EIA-232-D interface
SIM06 -- DCE with EIA-232-D interface
SIM07 -- DTE with EIA-530 interface
SIM08 -- DCE with EIA-530 interface

Port B is routed, via the SIM, to:

- ❏ The DB-25 connector marked SERIAL PORT 2 on the front panel of the **MVME162**. SERIAL PORT 2 will be an EIA-232-D DCE or DTE port, or an EIA-530 DCE or DTE port, depending upon which SIM is installed.

  NOTE: This port can be connected to the TX and RX clocks which may be present on the DB-25 connector. These connections are made via jumper header J12 on the MVME162 board. (The TxC and RxC clock lines are available at the MVME712M SERIAL PORT 4 via header J15, but are not available on the other MVME712X transition modules.)

- One of the following output connectors on the MVME712X transition module:

  MVME712M: The DB-25 connector marked SERIAL PORT 4 on the front panel. SERIAL PORT 4 can be configured as an EIA-232-D DTE or DCE port, via the jumper headers J18 and J19 on the MVME712M.

  MVME712A, AM, -12, or -13: The DB-9 connector marked SERIAL PORT 4 on the front panel. SERIAL PORT 4 is hard-wired as an EIA-232-D DTE port.

Figure 2-3 (sheets 1 through 6) in Chapter 2 illustrates the six configurations available for Port B when the MVME162 is used with an MVME712M. Note that the port configurations shown in Figure 2-3 sheets 5 and 6 are not recommended for synchronous applications because of the incorrect clock direction. Figure 2-4 (sheets 1 and 2) shows an MVME162 with the two configurations available with EIA-530 SIMs. Figure 2-5 (sheets 1 through 4) shows the four configurations available for Port B when the MVME162 is used with an MVME712A / AM / -12 / -13.

Caution **Do not simultaneously connect serial data devices to the equivalent ports on the MVME712 series transition module and the MVME162 front panel. This could result in simultaneous transmission of conflicting data.**

Caution **Do not connect peripheral devices to Port 1, Port 3, or the Centronics printer port on the MVME712X module.**

Caution **When using an EIA-530 SIM, do not connect the MVME162 to an MVME712X board. The EIA-530 signals are not supported by the P2 adapter and the transition boards.**

IndustryPack (IP) Interfaces

The IPIC ASIC on the MVME162 supports four IndustryPack (IP) interfaces: these are accessible from the front panel. Refer to the IPIC Programming Model in the *MVME162 Embedded Controller Programmer's Reference Guide* for details of the IP interface. Refer to the *MVME162 Embedded Controller Support Information* manual for the pin assignments of the IP connectors.

Optional LAN Ethernet Interface

The MVME162 uses the 82596CA to implement the Ethernet transceiver interface. The 82596CA accesses local RAM using DMA operations to perform its normal functions. Because the 82596CA has small internal buffers and the

VMEbus has an undefined latency period, buffer overrun may occur if the DMA is programmed to access the VMEbus. Therefore, the 82596CA should not be programmed to access the VMEbus.

Every MVME162 that has the Ethernet interface 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 MVME162 has a different value for XXXXX).

Each board 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 MVME162 debugger has the capability to retrieve or set the Ethernet address.

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

The Ethernet transceiver interface is located on the MVME162 main board, and the industry DB15 standard connector is located on the MVME712X transition board.

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

Optional SCSI Interface

The MVME162 may provide 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 MCchip ASIC. Refer to the 53C710 user's guide for detailed programming information.

SCSI Termination

The system configurer must ensure that the SCSI bus is properly terminated at both ends. On the MVME162, sockets are provided for the terminators on the P2 adapter board or the LCP2 adapter board. If the SCSI bus ends at the adapter board, then termination resistors must be installed on the adapter board. +5V power to the SCSI bus TERM power line and termination resistors is provided through a fuse located on the adapter board.

Local Resources

The MVME162 includes many resources for the local processor. These include tick timers, software-programmable hardware interrupts, watchdog timer, and local bus timeout.

Programmable Tick Timers

Six 32-bit programmable tick timers with 1 µs resolution are provided, two in the VMEchip2 and four in the MCchip. The tick timers can be programmed to generate periodic interrupts to the processor. Refer to the VMEchip2 and MCchip in the *MVME162 Embedded Controller Programmer's Reference Guide* for detailed programming information.

Watchdog Timer

A watchdog timer function is provided in the VMEchip2 and the MCchip. When the watchdog timer is enabled, it must be reset by software within the programmed time or it times out. The watchdog timer can be programmed to generate a SYSRESET signal, local reset signal, or board fail signal if it times out. Refer to the VMEchip2 and the MCchip in the *MVME162 Embedded Controller Programmer's Reference Guide* for detailed programming information.

The watchdog timer logic is duplicated in the VMEchip2 and MCchip ASICs. Because the watchdog timer function in the VMEchip2 is a superset of that function in the MCchip (system reset function), the timer in the VMEchip2 is used in all cases except for the version of the MVME162 which does not include the VMEbus interface ("No VMEbus Interface" option).

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 MVME162 provides a timeout function in the VMEchip2 and the MCchip 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 infinity. The local bus timer does not operate during VMEbus bound cycles. VMEbus bound cycles are timed by the VMEbus access timer and the VMEbus global timer.

The access timer logic is duplicated in the VMEchip2 and MCchip ASICs. Because the local bus timer in the VMEchip2 can detect an offboard access and the MCchip local bus timer cannot, the timer in the VMEchip2 is used in all cases except for the version of the MVME162 which does not include the VMEbus interface ("No-VMEbus-Interface option").

Local Bus Arbiter

The local bus arbiter implements a fixed priority which is described in the following table.

Table 1-2. Local Bus Arbitration Priority

Device	Priority	Note
LAN	0	Highest
SCSI	1	...
VMEbus	2	Next Lowest
MC68xx040	3	Lowest

Connectors

The MVME162 has two 96-position DIN connectors: P1 and P2. P1 rows A, B, C, and P2 row B provide the VMEbus interconnection. P2 rows A and C provide the connection to the SCSI bus, serial ports, and Ethernet. The MVME162 has a 20-pin connector J4 mounted behind the front panel. When the MVME162 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 serial ports on the MVME162 are also connected to two 25-pin DB-25 female connectors J9 and J15 on the front panel. The four IPs connect to the MVME162 by four pairs of 50-pin connectors. Four 50-pin connectors behind the front panel are for external connections to IP signals. The memory chip mezzanine board is plugged into two 40-pin connectors.

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), and 2) the mapping of onboard resources as viewed by VMEbus Masters (VMEbus memory map).

The memory and I/O maps which are described in the following tables are correct for all local bus masters. There is some address translation capability in the VMEchip2. This allows multiple MVME162s on the same VMEbus with different virtual local bus maps as viewed by different VMEbus masters.

Local Bus Memory Map

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

Normal Address Range

The memory map of devices that respond to the normal address range is shown in the following tables. The normal address range is defined by the Transfer Type (TT) signals on the local bus. On the MVME162, Transfer Types 0, 1, and 2 define the normal address range. Table 1-3 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 MC68xx040 MMU. The onboard I/O space must be marked cache inhibit and serialized in its page table. Table 1-4 further defines the map for the local I/O devices.

Table 1-3. Local Bus Memory Map

Address Range	Devices Accessed	Port Width	Size	Software Cache Inhibit	Note(s)
Programmable	DRAM on board	D32	1MB-4MB	N	2
Programmable	SRAM	D32	128KB-2MB	N	2
Programmable	VMEbus A32/A24	D32/D16	--	?	4
Programmable	IP a Memory	D32-D8	64KB-8MB	?	2, 4
Programmable	IP b Memory	D32-D8	64KB-8MB	?	2, 4
Programmable	IP c Memory	D32-D8	64KB-8MB	?	2, 4
Programmable	IP d Memory	D32-D8	64KB-8MB	?	2, 4
$FF800000 - $FF9FFFFF	Flash/PROM	D32	2MB	N	1, 5
$FFA00000 - $FFBFFFFF	PROM/Flash	D32	2MB	N	6
$FFC00000 - $FFCFFFFF	not decoded	--	1MB	N	7
$FFD00000 - $FFDFFFFF	not decoded	--	1MB	N	7
$FFE00000 - $FFE7FFFF	SRAM default	D32	512KB	N	--
$FFE80000 - $FFEFFFFF	not decoded	--	512KB	N	7
$FFF00000 - $FFFEFFFF	Local I/O	D32-D8	878KB	Y	3
$FFFF0000 - $FFFFFFFF	VMEbus A16	D32/D16	64KB	?	2, 4

NOTES:

1. Reset enables the decoder for this space of the memory map so that it will decode address spaces $FF800000 - $FF9FFFFF and $00000000 - $003FFFFF. The decode at 0 must be disabled in the MCchip before DRAM is enabled. DRAM is enabled with the DRAM Control Register at address $FFF42048, bit 24. PROM / Flash is disabled at the low address space with PROM Control Register at address $FFF42040, bit 20.
2. This area is user-programmable. The DRAM and SRAM decoder is programmed in the MCchip, the local-to-VMEbus decoders are programmed in the VMEchip2, and the IP memory space is programmed in the IPIC.
3. Size is approximate.
4. Cache inhibit depends on devices in area mapped.
5. The PROM and Flash are sized by the MCchip ASIC from an 8-bit private bus to the 32-bit MPU local bus. Because the device size is less than the allocated memory map size for some entries, the device contents repeat for those entries.
   If jumper GPI3 is installed, the Flash device is accessed. If GPI3 is not installed, the PROM is accessed.
6. The Flash and PROM are sized by the MCchip ASIC from an 8-bit private bus to the 32-bit MPU local bus. Because the device size is less than the allocated memory map size for some entries, the device contents repeat for those entries.
   If jumper GPI3 is installed, the PROM is accessed. If GPI3 is not installed, the Flash device is accessed.
7. These areas are not decoded unless one of the programmable decoders are initialized to decode this space. If they are not decoded, an access to this address range will generate a local bus timeout. The local bus timer must be enabled.

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

Table 1-4. Local Bus I/O Devices Memory Map

Address Range	Device	Port Width	Size	Note(s)
$FFF00000 - $FFF3FFFF	reserved	--	256KB	4
$FFF40000 - $FFF400FF	VMEchip2 (LCSR)	D32	256B	1, 3
$FFF40100 - $FFF401FF	VMEchip2 (GCSR) registers	D32-D8	256B	1, 3
$FFF40200 - $FFF40FFF	reserved	--	3.5KB	4, 5
$FFF41000 - $FFF41FFF	reserved	--	4KB	4
$FFF42000 - $FFF42FFF	MCchip	D32-D8	4KB	1
$FFF44300 - $FFF44FFF	reserved	--	8KB	4
$FFF45000 - $FFF45FFF	SCC (Z85230)	D8	4KB	1, 2
$FFF46000 - $FFF46FFF	LAN (82596CA)	D32	4KB	1, 6
$FFF47000 - $FFF47FFF	SCSI (53C710)	D32-D8	4KB	1
$FFF48000 - $FFF57FFF	reserved	--	64KB	4
$FFF58000 - $FFF5807F	IPIC IP a I/O	D16	128B	1
$FFF58080 - $FFF580FF	IPIC IP a ID	D16	128B	1
$FFF58100 - $FFF5817F	IPIC IP b I/O	D16	128B	1
$FFF58180 - $FFF581FF	IPIC IP b ID Read	D16	128B	1
$FFF58200 - $FFF5827F	IPIC IP c I/O	D16	128B	1
$FFF58280 - $FFF582FF	IPIC IP c ID	D16	128B	1
$FFF58300 - $FFF5837F	IPIC IP d I/O	D16	128B	1
$FFF58380 - $FFF583FF	IPIC IP d ID Read	D16	128B	1
$FFF58400 - $FFF584FF	IPIC IP ab I/O	D32-D16	256B	1
$FFF58500 - $FFF585FF	IPIC IP cd I/O	D32-D16	256B	1
$FFF58600 - $FFF586FF	IPIC IP ab I/O repeated	D32-D16	256B	1
$FFF58700 - $FFF587FF	IPIC IP cd I/O repeated	D32-D16	256B	1
$FFF58800 - $FFF5887F	reserved	--	128B	1
$FFF58880 - $FFF588FF	reserved	--	128B	1
$FFF58900 - $FFF5897F	reserved	--	128B	1
$FFF58980 - $FFF589FF	reserved	--	128B	1
$FFF58A00 - $FFF58A7F	reserved	--	128B	1
$FFF58A80 - $FFF58AFF	reserved	--	128B	1
$FFF58B00 - $FFF58B7F	reserved	--	128B	1
$FFF58B80 - $FFF58BFF	reserved	--	128B	1
$FFF58C00 - $FFF58CFF	reserved	--	256B	1
$FFF58D00 - $FFF58DFF	reserved	--	256B	1
$FFF58E00 - $FFF58EFF	reserved	--	256B	1
$FFF58F00 - $FFF58FFF	reserved	--	256B	1
$FFFBC000 - $FFFBC01F	IPIC registers	D32-D8	2KB	1
$FFFBC800 - $FFFBC81F	reserved	--	2KB	1
$FFFBD000 - $FFFBFFFF	reserved	--	12KB	4
$FFFC0000 - $FFFC7FFF	MK48T08 (BBRAM, TOD clock)	D32-D8	32KB	1
$FFFC8000 - $FFFCBFFF	MK48T08 & disable Flash writes	D32-D8	16KB	1, 7
$FFFCC000 - $FFFCFFFF	MK48T08 & enable Flash writes	D32-D8	16KB	1, 7
$FFFD0000 - $FFFEFFFF	reserved	--	128KB	4

NOTES:

1. For a complete description of the register bits, refer to the *MVME162 Embedded Controller Programmer's Reference Guide* or to the data sheet for the specific chip.
2. The SCC is an 8-bit device located on an MCchip private data bus. Byte access is required.
3. Writes to the LCSR in the VMEchip2 must be 32 bits. LCSR writes of 8 or 16 bits terminate with a TEA signal. Writes to the GCSR may be 8, 16 or 32 bits. Reads to the LCSR and GCSR may be 8, 16 or 32 bits. Byte reads should be used to read the interrupt vector.
4. This area does not return an acknowledge signal. If the local bus timer is enabled, the access times out and is terminated by a TEA signal.
5. Size is approximate.
6. Port commands to the 82596CA must be written as two 16-bit writes: upper word first and lower word second.
7. Refer to the Flash and PROM Interface section in the MCchip description in the *MVME162 Embedded Controller Programmer's Reference Guide*.

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 MVME162 responds to.

VMEbus Short I/O Memory Map

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

HARDWARE PREPARATION AND INSTALLATION

2

Introduction

This chapter provides unpacking instructions, hardware preparation, and installation instructions for the MVME162 Embedded Controller. Hardware preparation for the MVME712 series transition modules is provided in separate manuals. 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 MVME162, certain option modifications may be necessary before installation. MVME162 provides software control for most of these options. Some options can not be performed in software, so are performed by installing or removing header jumpers or interface modules. Most other modifications are performed by setting bits in control registers after MVME162 has been installed in a system. (For more information on the MVME162 registers refer to the *MVME162 Embedded Controller Programmer's Reference Guide* listed in *Related Documentation* in Chapter 1.)

The locations of the switches, jumper headers, connectors, and LEDs on the MVME162 are illustrated in Figure 2-1. MVME162 has been factory tested and is shipped with the factory jumper settings described in the following sections.

MVME162 operates with its required and factory-installed Debug Monitor, MVME162Bug (162Bug), with these factory jumper settings. Manually configurable items include:

- ❏ SIM selection for serial port B configuration (J10)
- ❏ System controller selection (J1)
- ❏ Synchronous clock selection (J11) for Serial Port 1/Console
- ❏ Synchronous clock selection (J12) for Serial Port 2
- ❏ SRAM backup power source selection (J20)
- ❏ EPROM size selection (J21)
- ❏ General-purpose readable register configuration (J22)

SIM Selection

Port B of the MVME162's Z85230 serial communications controller is configurable via a serial interface module (SIM) which is installed at connector J10 on the MVME162 board. Four serial interface modules are available:

- ❏ EIA-232-D (DCE and DTE)
- ❏ EIA-530 (DCE and DTE)

You can change Port B from an EIA-232-D to an EIA-530 interface (or vice-versa) by mounting the appropriate serial interface module. Port B is routed (via the SIM at J10) to the 25-pin DB25 front panel connector marked SERIAL PORT 2.

For the location of SIM connector J10 on the MVME162, refer to Figure 2-1. Figure 2-2 illustrates the secondary side (bottom) of a serial interface module, showing the J1 connector which plugs into SIM connector J10 on the MVME162. Figure 2-3 (sheets 3-6) and Figure 2-4 illustrate the six configurations available for Port B.

For the part numbers of the serial interface modules, refer to Table 2-1. The part numbers are ordinarily printed on the primary side (top) of the SIMs, but may be found on the secondary side in some versions.

If you need to replace an existing serial interface module with a SIM of another type, go to *Removal of Existing SIM* below. If there is no SIM on the main board, skip to *Installation of New SIM*.

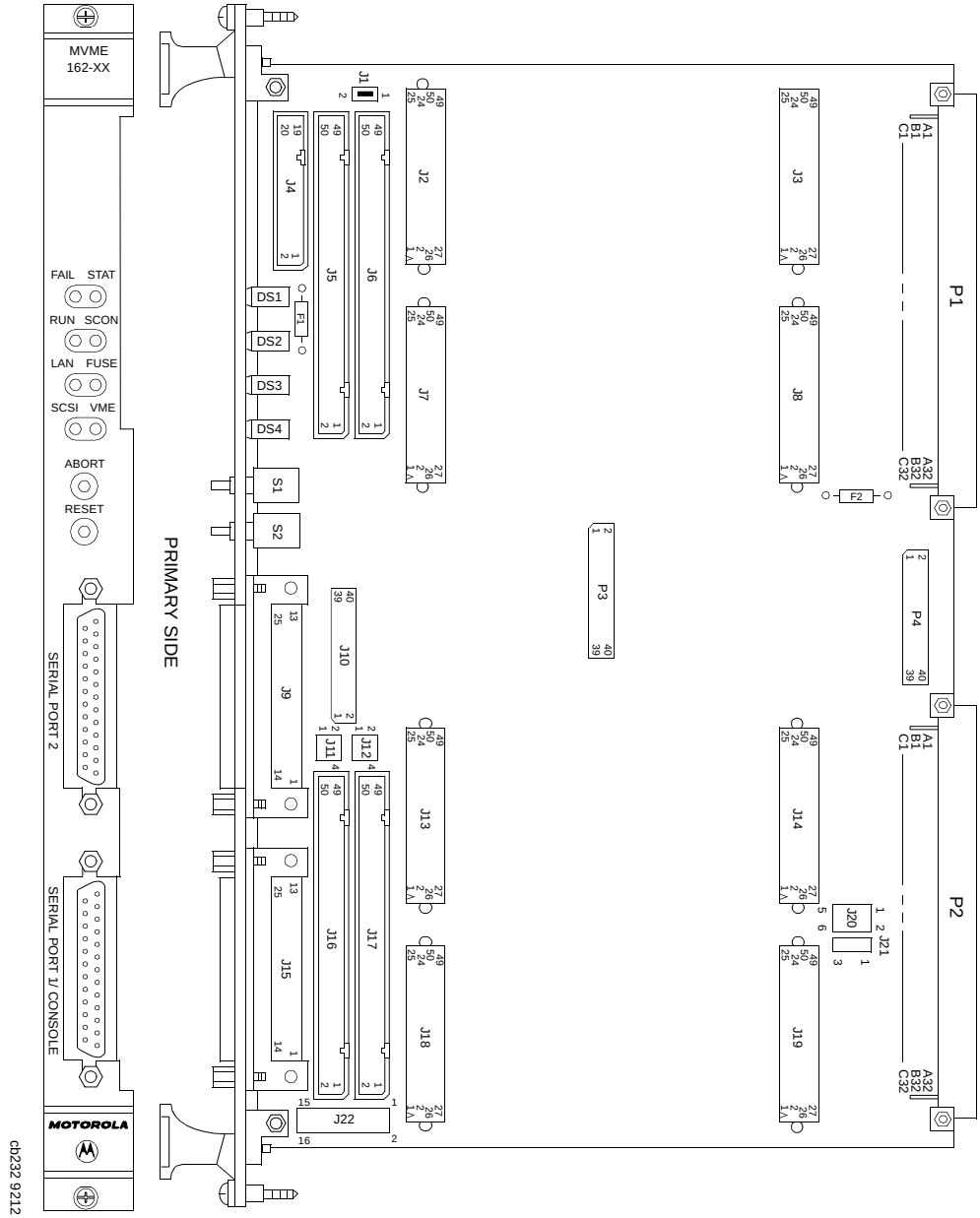


Figure 2-1. MVME162 Switches, Headers, Connectors, Fuses, and LEDs

Table 2-1. Serial Interface Module Part Numbers

EIA Standard	Configuration	Part Number	Model Number
EIA-232-D	DTE	01-W3846B	SIM05
	DCE	01-W3865B	SIM06
EIA-530	DTE	01-W3868B	SIM07
	DCE	01-W3867B	SIM08

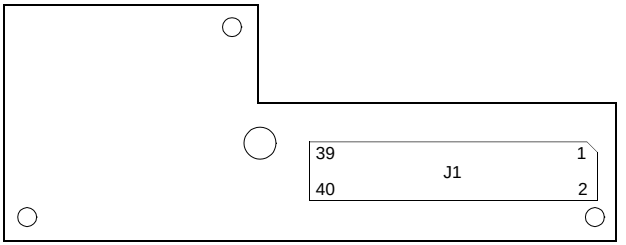


Figure 2-2. Serial Interface Module, Connector Side

Removal of Existing SIM

1. Each serial interface module is retained by two 4-40 x 3/16 " Phillips-head screws in opposite corners. Remove the two screws and store them in a safe place for later use.
2. Grasp opposite sides of the SIM and gently lift straight up.

Caution **Avoid lifting the SIM by one side only, as the connector can be damaged on the SIM or the main board.**

3. Place the SIM in a static-safe container for possible reuse.

Installation of New SIM

1. Observe the orientation of the connector keys on SIM connector J1 and MVME162 connector J10. Turn the SIM so that the keys line up and place it gently on connector J10, aligning the mounting holes at the SIM corners with the matching standoffs on the MVME162.
2. Gently press the top of the SIM to seat it on the connector. If the SIM does not seat with gentle pressure, recheck the orientation. If the SIM connector is oriented incorrectly, the mounting holes will not line up with the standoffs.

Caution **Do not attempt to force the SIM on if it is oriented incorrectly.**

3. Place the two 4-40 x 3/16" Phillips-head screws that you previously removed (or that were supplied with the new SIM) into the two opposite-corner mounting holes. Screw them into the standoffs but do not overtighten them.

The signal relationships and signal connections in the various serial configurations available for ports A and B are illustrated in Figures 2-3 and 2-4.

System Controller Select Header (J1)

The MVME162 is factory-configured as a VMEbus system controller (i.e., a jumper is installed across pins 1 and 2 of header J1). Remove the J1 jumper if the MVME162 is not to be the system controller. Note that when the MVME162 is functioning as system controller, the SCON LED is turned on.

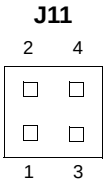
Note **For MVME162s without the optional VMEbus interface (i.e., no VMEchip2), the jumper may be installed or removed without affecting normal operation.**

System Controller (factory configuration)

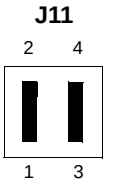
Not System Controller

Synchronous Clock Select Header (J11) for Serial Port 1/Console

The MVME162 is shipped from the factory with the SERIAL PORT 1/CONSOLE header configured for asynchronous communications (i.e., jumpers removed). To select synchronous communications for the SERIAL PORT 1/CONSOLE connection, install jumpers across pins 1 and 2 and pins 3 and 4.



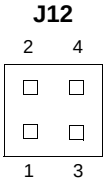
Internal Clock (factory configuration)



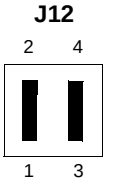
External Clock

Clock Select Header (J12) for Serial Port 2

The MVME162 is shipped from the factory with the SERIAL PORT 2 header configured for asynchronous communications (i.e., jumpers removed). To select synchronous communications for the SERIAL PORT 2 connection, install jumpers across pins 1 and 2 and pins 3 and 4.



Internal Clock (factory configuration)



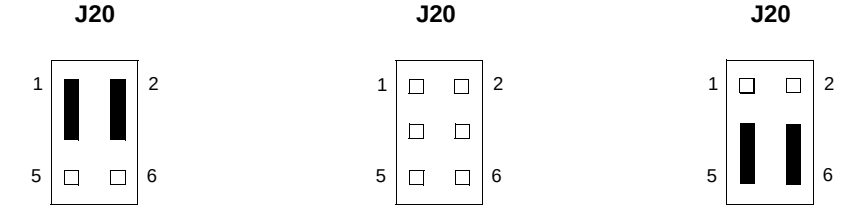
External Clock

SRAM Battery Backup Source Select Header (J20)

The MVME162 is factory-configured to use VMEbus +5V Standby power as a backup power source for the SRAM (i.e., jumpers are installed across pins 1 and 3 and 2 and 4). To select the onboard battery as the backup power source, install the jumpers across pins 3 and 5 and 4 and 6.

Note **For MVME162s without optional VMEbus interface (i.e., without VMEchip2 ASIC), you must select the onboard battery for the backup power source.**

Caution **Removing all jumpers may temporarily disable the SRAM. Do not remove all jumpers from J20, except for storage**



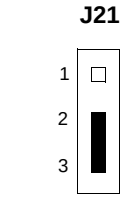
VMEbus +5V STBY
(Factory configuration)

Backup Power Disabled
(For storage only)

Onboard Battery

EPROM Size Select Header (J21)

The MVME162 is factory-configured for a 4Mbit EPROM (i.e., a jumper is installed across pins 2 and 3). This is the only size currently available; if a larger PROM becomes available, this jumper will allow it to be selected.



4Mbit EPROM
(Factory configuration)

2

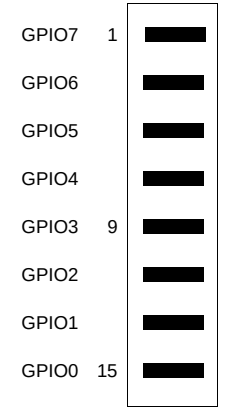
General Purpose Readable Jumpers Header (J22)

Header J22 provides eight readable jumpers. These jumpers are read as a register (at $FFF4202D) in the MCchip LCSR (local control/status register). The bit values are read as a zero when the jumper is installed and as a one when the jumper is removed.

If the MVME162BUG firmware is installed, four jumpers are user-definable (pins 1-2, 3-4, 5-6, 7-8). If the MVME162BUG firmware is not installed, seven jumpers are user-definable (pins 1-2, 3- 4, 5-6, 7-8, 11-12, 13-14, 15-16).

Note **Pins 9-10 (GPIO3) are reserved to select either the Flash memory map (jumper installed) or the EPROM memory map (jumper removed). They are not user-definable.**

The MVME162 is shipped from the factory with J22 set to all zeros (jumpers on all pins).



J22	162BUG INSTALLED	USER CODE INSTALLED
GPIO7 1 2	USER-DEFINABLE	USER-DEFINABLE
GPIO6	USER-DEFINABLE	USER-DEFINABLE
GPIO5	USER-DEFINABLE	USER-DEFINABLE
GPIO4	USER-DEFINABLE	USER-DEFINABLE
GPIO3 9 10	IN=FLASH; OUT=EPROM	IN=FLASH; OUT=EPROM
GPIO2	REFER TO 162BUG MANUAL	USER-DEFINABLE
GPIO1	REFER TO 162BUG MANUAL	USER-DEFINABLE
GPIO0 15 16	REFER TO 162BUG MANUAL	USER-DEFINABLE

EPROMs Selected (factory configuration)

Installation Instructions

The following sections discuss the installation of IndustryPacks (IPs) on the MVME162, the installation of the MVME162 into a VME chassis, and the system considerations relevant to the installation. Before installing IndustryPacks, ensure that the serial ports and all header jumpers are configured as desired.

IP Installation on the MVME162

Up to four IndustryPack (IP) modules may be installed on the MVME162. Install the IPs on the MVME162 as follows:

1. Each IP has two 50-pin connectors that plug into two corresponding 50-pin connectors on the MVME162: J2 / J3, J7 / J8, J13 / J14, J18 / J19. See Figure 2-1 for the MVME162 connector locations.
   - Orient the IP(s) so that the tapered connector shells mate properly. Plug IP_a into connectors J2 and J3; plug IP_b into J7 and J8. Plug IP_c into J13 and J14; plug IP_d into J18 and J19. If a double-sized IP is used, plug IP_ab into J2, J3, J7, and J8; plug IP_cd into J13, J14, J18, and J19.
2. Four additional 50-pin connectors (J6, J5, J17, and J16) are provided behind the MVME162 front panel for external cabling connections to the IP modules. There is a one-to-one correspondence between the signals on the cabling connectors and the signals on the associated IP connectors (i.e., J6 has the same IP_a signals as J2; J5 has the same IP_b signals as J7; J17 has the same IP_c signals as J13; and J16 has the same IP_d signals as J18.
   - Connect user-supplied 50-pin cables to J6, J5, J17, and J16 as needed. Because of the varying requirements for each different kind of IP, Motorola does not supply these cables.
   - Bring the IP cables out the narrow slots in the MVME162 front panel and attach them to the appropriate external equipment, depending on the nature of the particular IP(s).

MVME162 Module Installation

With EPROM, IndustryPack, and SIMs installed and headers properly configured, proceed as follows to install the MVME162 in the VME chassis:

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

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

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

2. Remove the chassis cover as instructed in the user's manual for the equipment.
3. Remove the filler panel from the card slot where you are going to install the MVME162.
   - If you intend to use the MVME162 as system controller, it must occupy the leftmost card slot (slot 1). The system controller must be in slot 1 to correctly initiate the bus-grant daisy-chain and to ensure proper operation of the IACK daisy-chain driver.
   - If you do not intend to use the MVME162 as system controller, it can occupy any unused double-height card slot.
4. Slide the MVME162 into the selected 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.
5. Secure the MVME162 in the chassis with the screws provided, making good contact with the transverse mounting rails to minimize RF emissions.
6. Install the MVME712 series transition module in the front or the rear of the VME chassis. (To install an MVME712M, which has a double-wide front panel, you may need to shift other modules in the chassis.)
7. On the chassis backplane, remove the INTERRUPT ACKNOWLEDGE (IACK) and BUS GRANT (BG) jumpers from the header for the card slot occupied by the MVME162.

8. Connect the P2 Adapter Board or LCP2 Adapter Board and cable(s) to MVME162 backplane connector P2. This provides a connection point for terminals or other peripherals at the EIA-232-D serial ports, SCSI ports, and LAN Ethernet port.

   For information on installing the P2 or LCP2 Adapter Board and the MVME712 series transition module(s), refer to the manuals listed in *Related Documentation* in Chapter 1 (the *MVME162 Embedded Controller Programmer's Reference Guide* provides some connection diagrams.)

9. Connect the appropriate cable(s) to the panel connectors for the EIA-232-D serial ports, SCSI port, and LAN Ethernet port.

   – Note that some cables are not provided with the MVME712 series module and must be made or purchased by the user. (Motorola recommends shielded cable for all peripheral connections to minimize radiation.)

10. Connect the peripheral(s) to the cable(s). Appendix A supplies detailed information on the EIA-232-D signals supported. Appendix B describes the Ethernet LAN (*Local Area Network*) port connections. Appendix C describes the SCSI (*Small Computer System Interface*) I/O bus connections.
11. Install any other required VMEmodules in the system.
12. Replace the chassis cover.
13. Connect the power cable to the AC power source and turn the equipment power ON.

System Considerations

The MVME162 draws power from VMEbus backplane connectors P1 and P2. P2 is also used for the upper 16 bits of data in 32-bit transfers, and for the upper 8 address lines used in extended addressing mode. The MVME162 may not function properly without its main board connected to VMEbus backplane connectors P1 and P2.

Whether MVME162 operates as VMEbus master or VMEbus slave, it is configured for 32 bits of address and 32 bits of data (A32/D32). However, it handles A16 or A24 devices in the address ranges indicated in Chapter 1. D8 and/or D16 devices in the system must be handled by the MC68040/MC68LC040 software. Refer to the memory maps in the *MVME162 Embedded Controller Programmer's Reference Guide*.)

The MVME162 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 MVME162Bug firmware. This may be changed via software to any other base address. Refer to *MVME162 Embedded Controller Programmer's Reference Guide* for more information.

If the MVME162 tries to access offboard resources in a nonexistent location and is not system controller, and if the system does not have a global bus timeout, the MVME162 waits forever for the VMEbus cycle to complete. This will cause the system to lock up. There is only one situation in which the system might lack this global bus timeout: when the MVME162 is not the system controller and there is no global bus timeout elsewhere in the system.

Multiple MVME162s may be installed in a single VME chassis. In general, hardware multiprocessor features are supported.

Note **If you are installing multiple MVME162s in an MVME945 chassis, do not install an MVME162 in slot 12. The height of the IP modules may cause clearance difficulties in that slot position.**

Other MPUs on the VMEbus can interrupt, disable, communicate with, and determine the operational status of the processor(s). One register of the GCSR (global control/status register) set includes four bits that function as location monitors to allow one MVME162 processor to broadcast a signal to any other MVME162 processors. All eight registers are accessible from any local processor as well as from the VMEbus.

The MVME162 provides +5 Vdc power to the remote LED/switch connector (J4) through a 1A fuse (F1) located near J4. Connector J4 is the interface for a remote control and indicator panel. If none of the LEDs light and the ABORT and RESET switches do not operate, check fuse F1.

The MVME162 provides +12 Vdc power to the Ethernet transceiver interface through a 1A fuse (F2) located near diode CR1. The FUSE LED lights to indicate that +12 Vdc is available. When the MVME712M module is used, the yellow DS1 LED on the MVME712M illuminates when LAN power is available, which indicates that the fuse is good. If the Ethernet transceiver fails to operate, check fuse F2.

The MVME162 provides SCSI terminator power through a 1A fuse (F1) located on the P2 Adapter Board or LCP2 Adapter Board. If the fuse is blown, the SCSI device(s) may function erratically or not at all. When the P2 Adapter Board is used with an MVME712M and the SCSI bus is connected to the MVME712M, the green DS2 LED on the MVME712M front panel illuminates when SCSI terminator power is available. If the green DS2 LED flickers during SCSI bus operation, check P2 Adapter Board fuse F1.

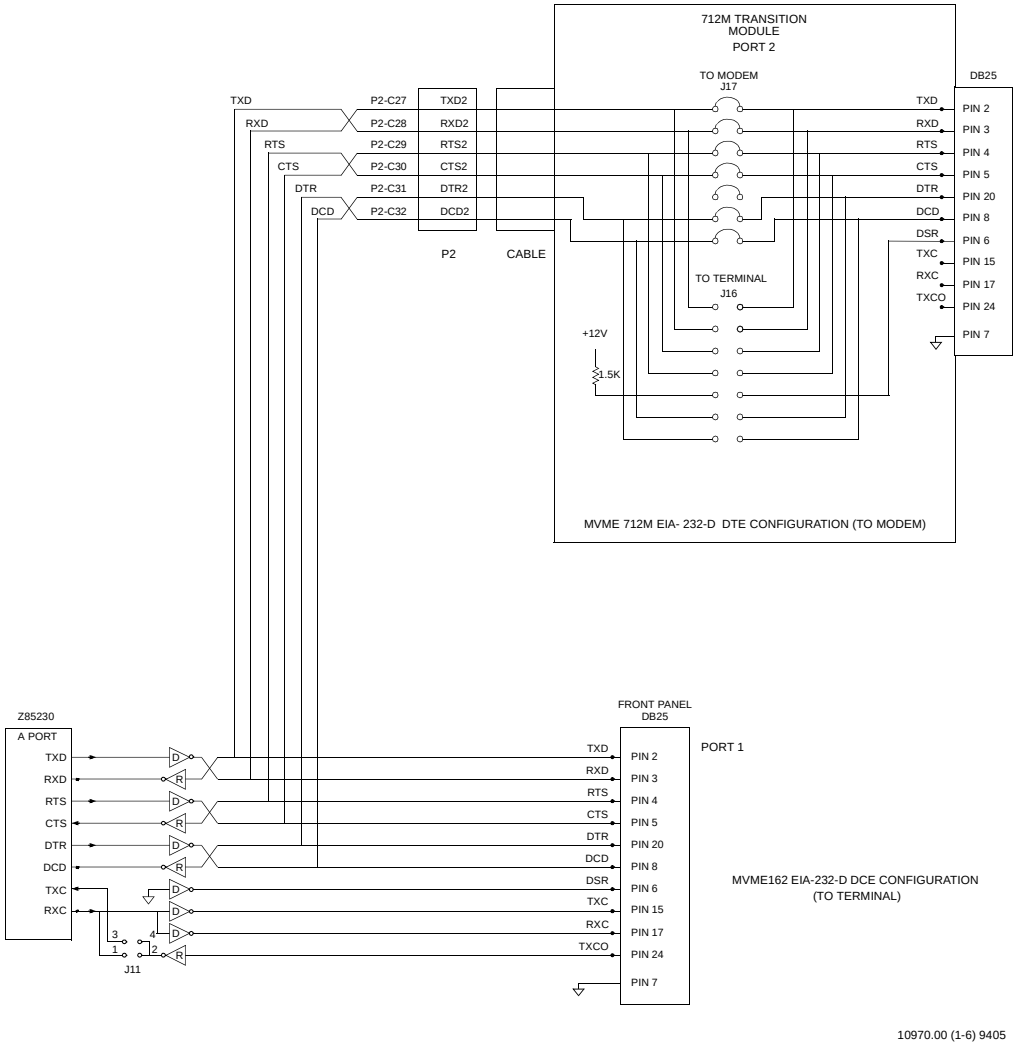


Figure 2-3. MVME162 EIA-232-D Connection Diagram, MVME712M (Sheet 1 of 6)

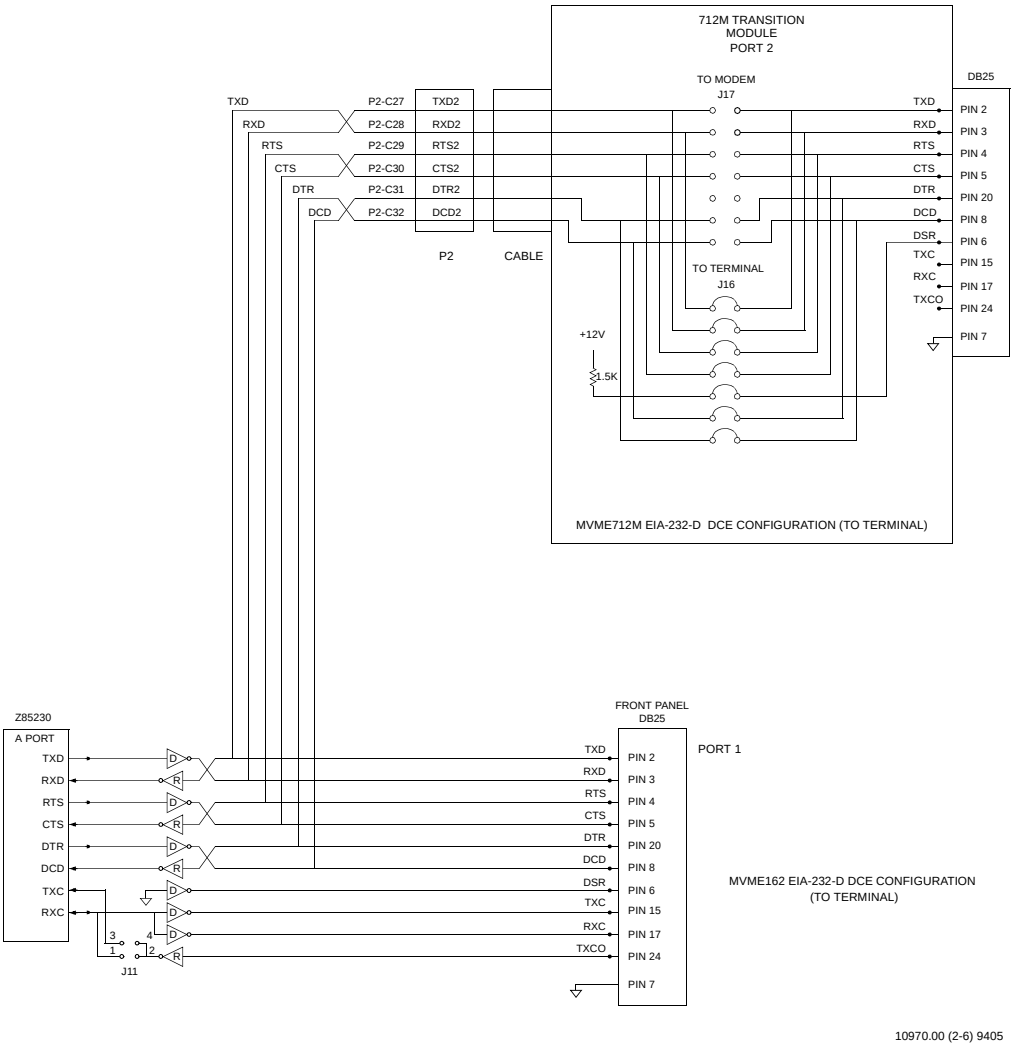


Figure 2-3. MVME162 EIA-232-D Connection Diagram, MVME712M (Sheet 2 of 6)

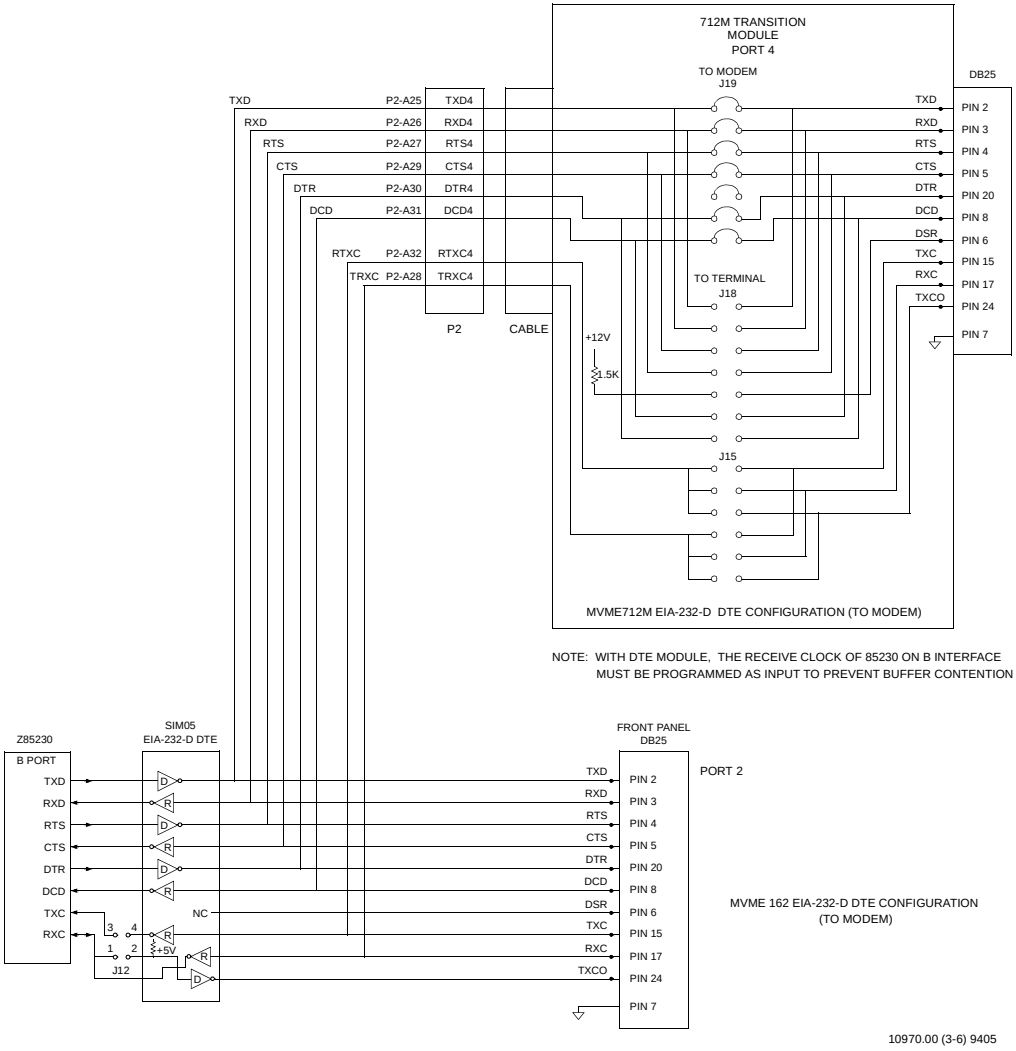


Figure 2-3. MVME162 EIA-232-D Connection Diagram, MVME712M (Sheet 3 of 6)

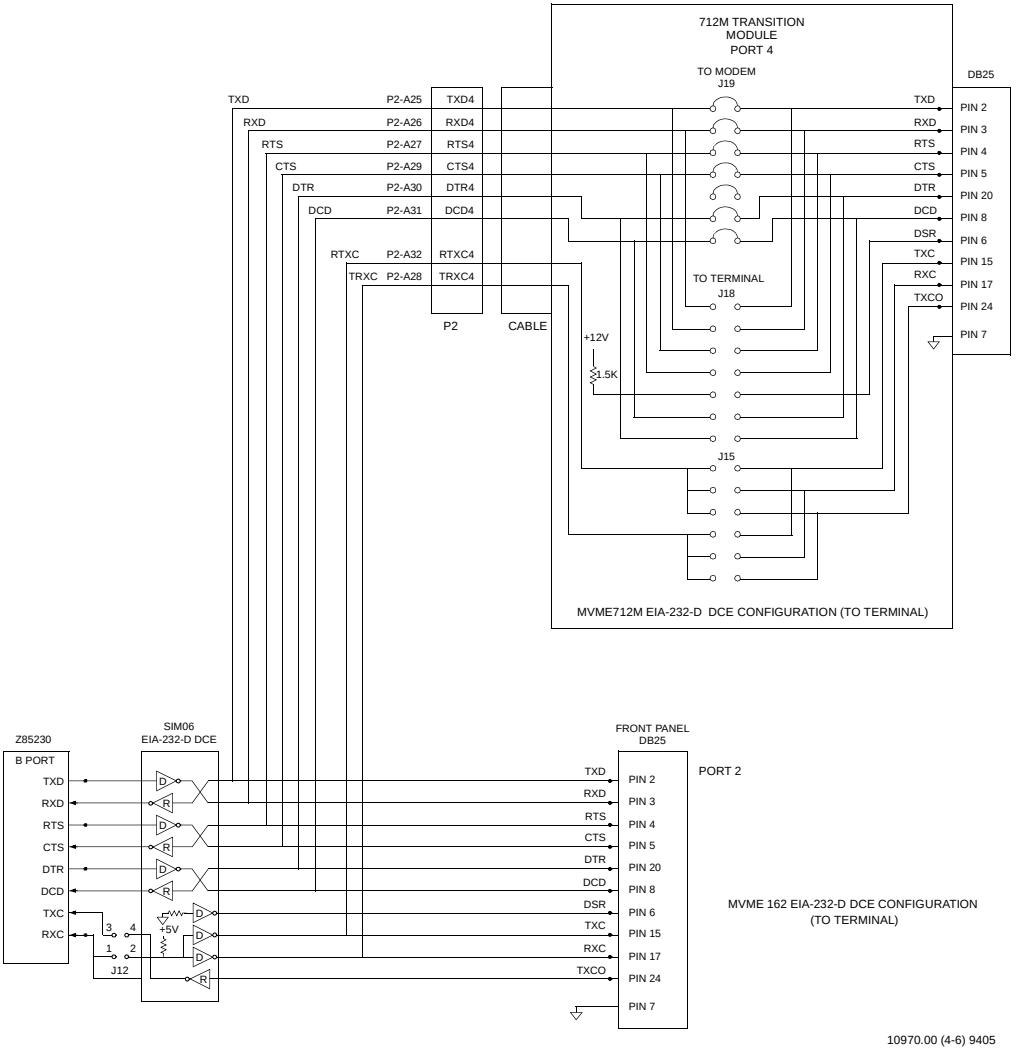


Figure 2-3. MVME162 EIA-232-D Connection Diagram, MVME712M (Sheet 4 of 6)

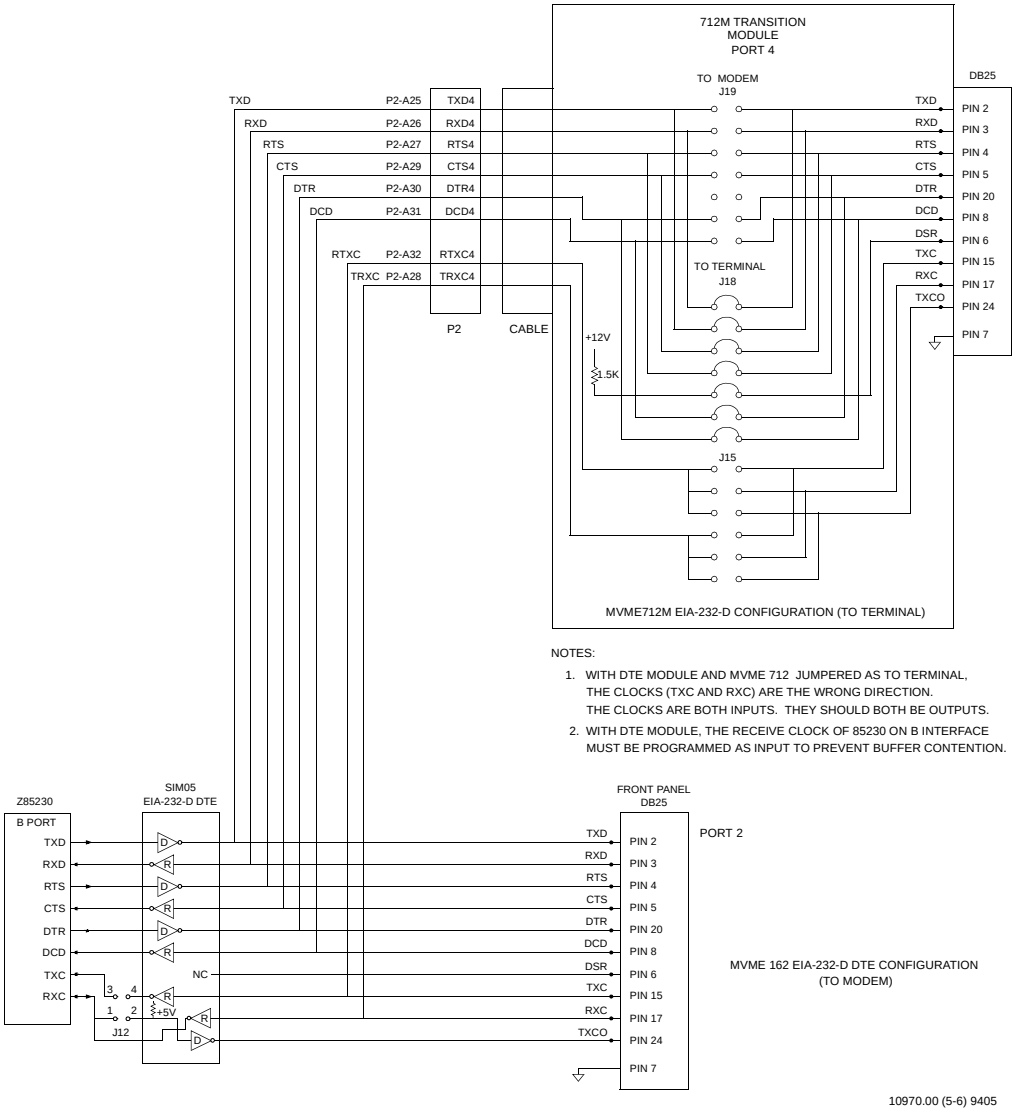


Figure 2-3. MVME162 EIA-232-D Connection Diagram, MVME712M (Sheet 5 of 6)

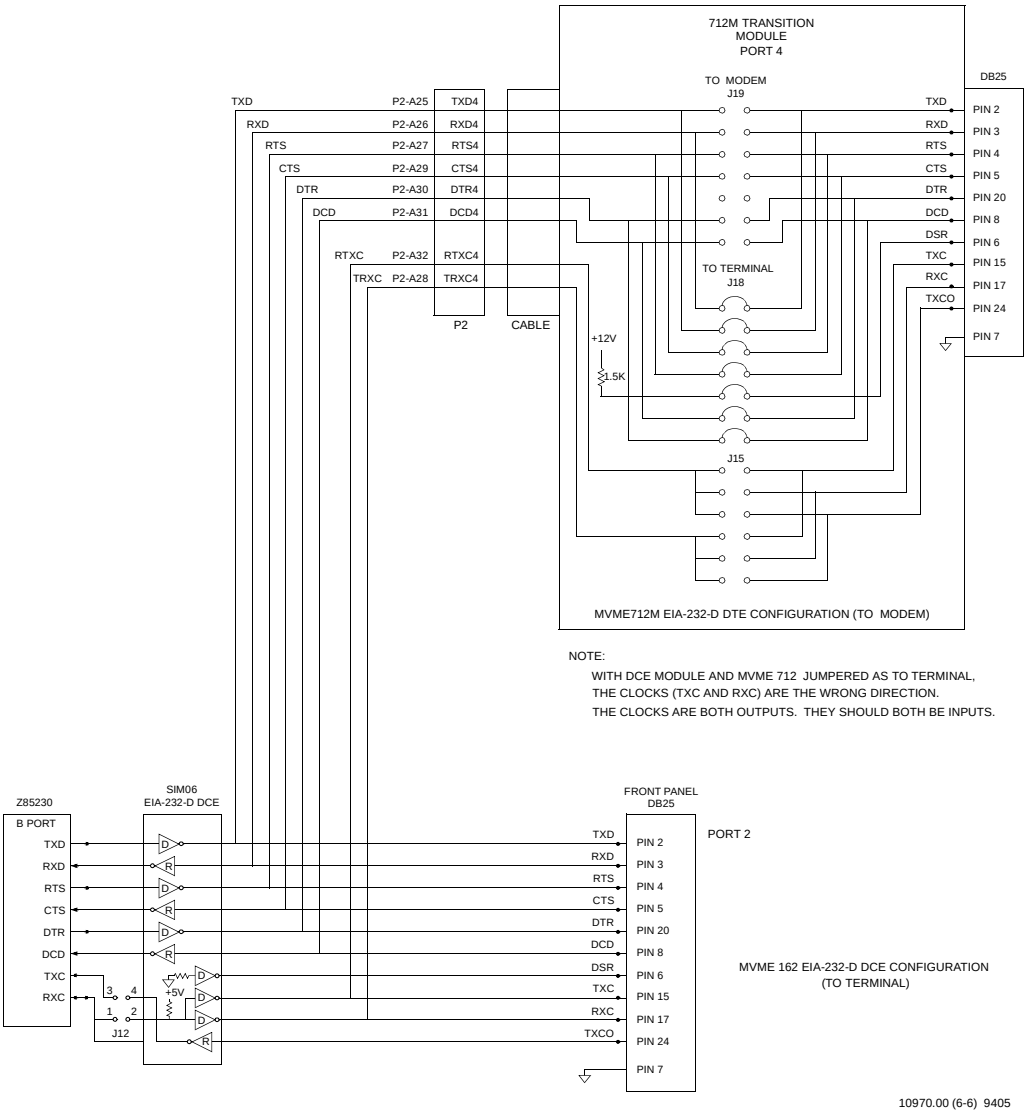
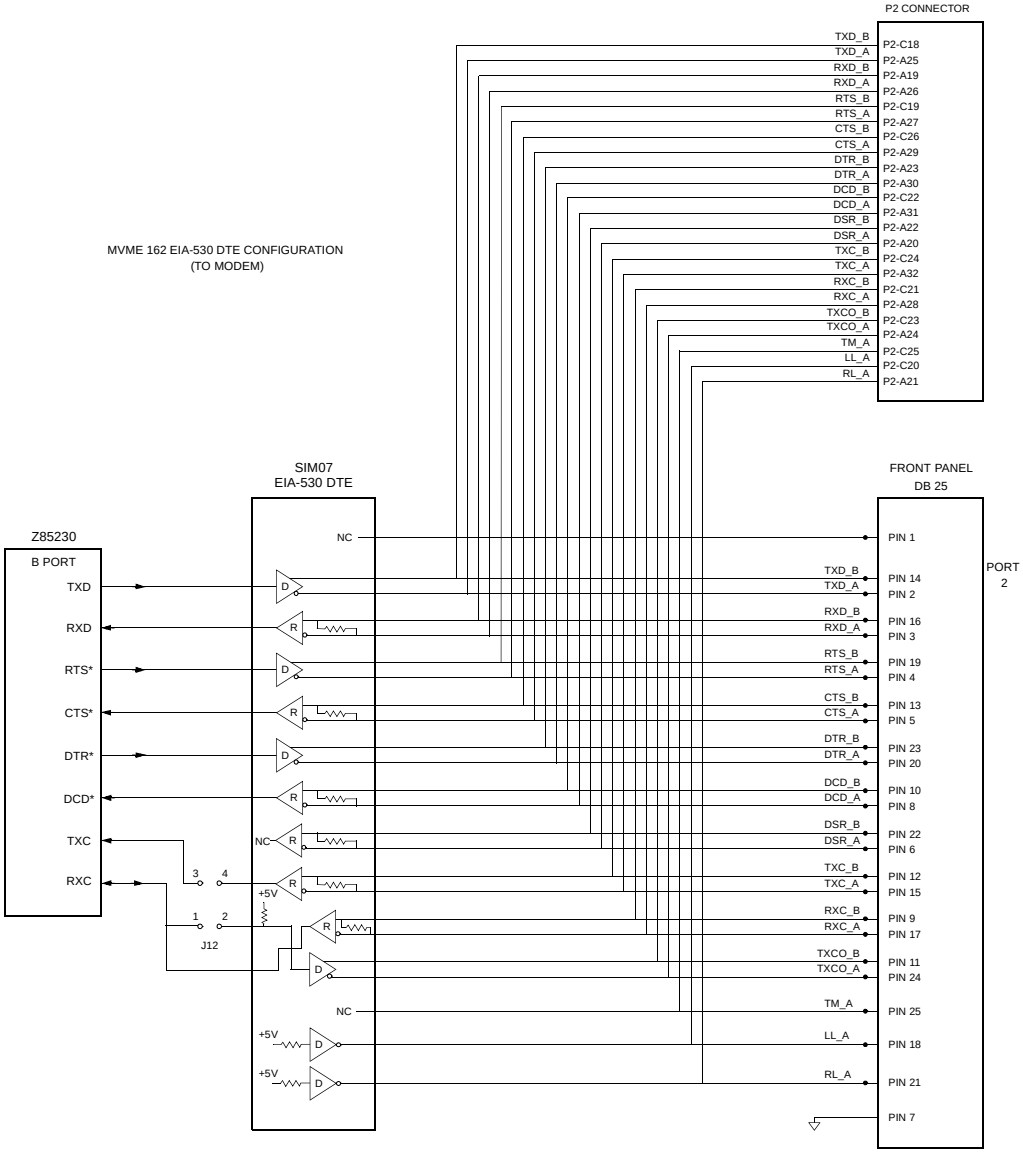


Figure 2-3. MVME162 EIA-232-D Connection Diagram, MVME712M (Sheet 6 of 6)



10971.00 (1-2) 9405

Figure 2-4. MVME162 EIA-530 Connection Diagram (Sheet 1 of 2)

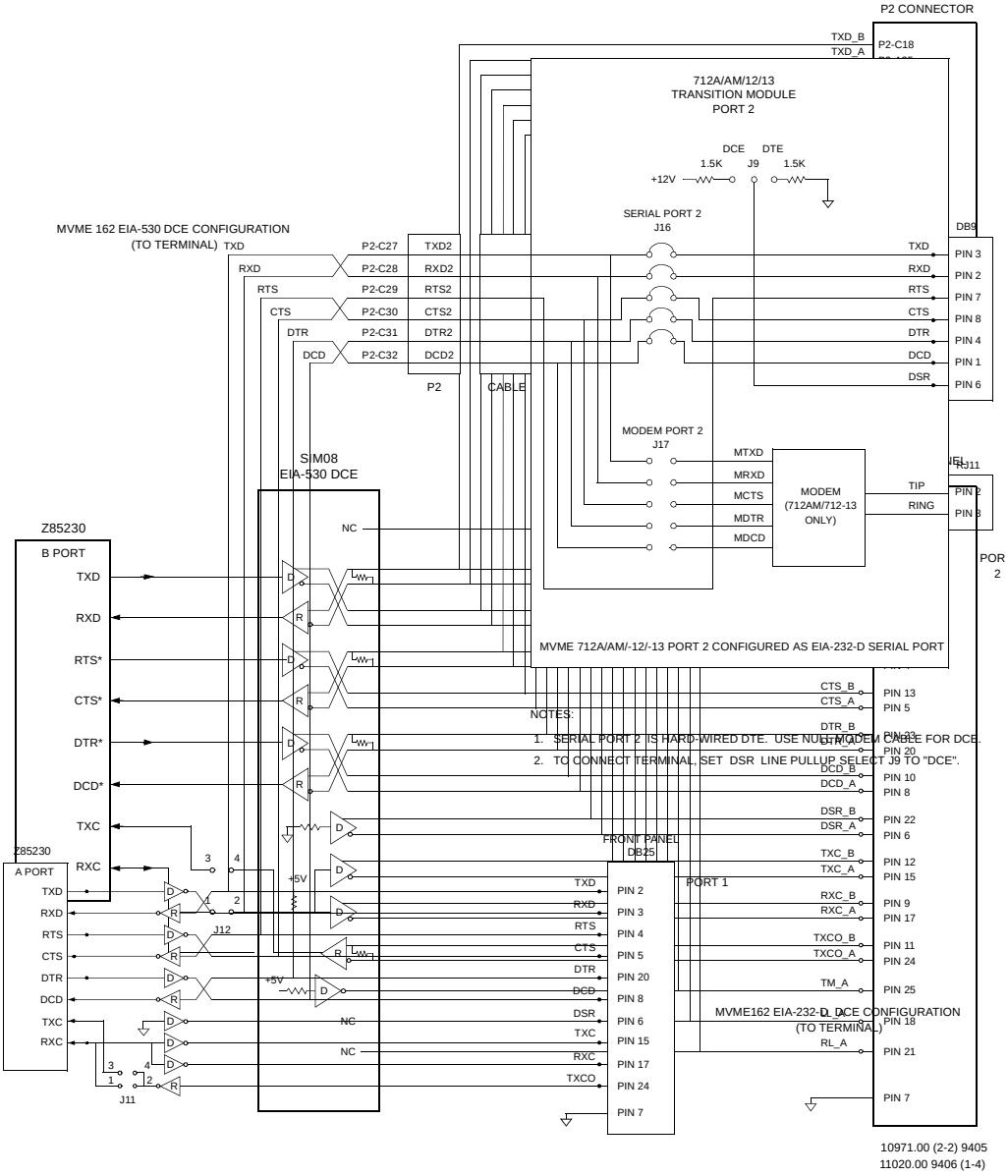


Figure 2-4. MVME162 EIA-530 Connection Diagram (Sheet 2 of 2)

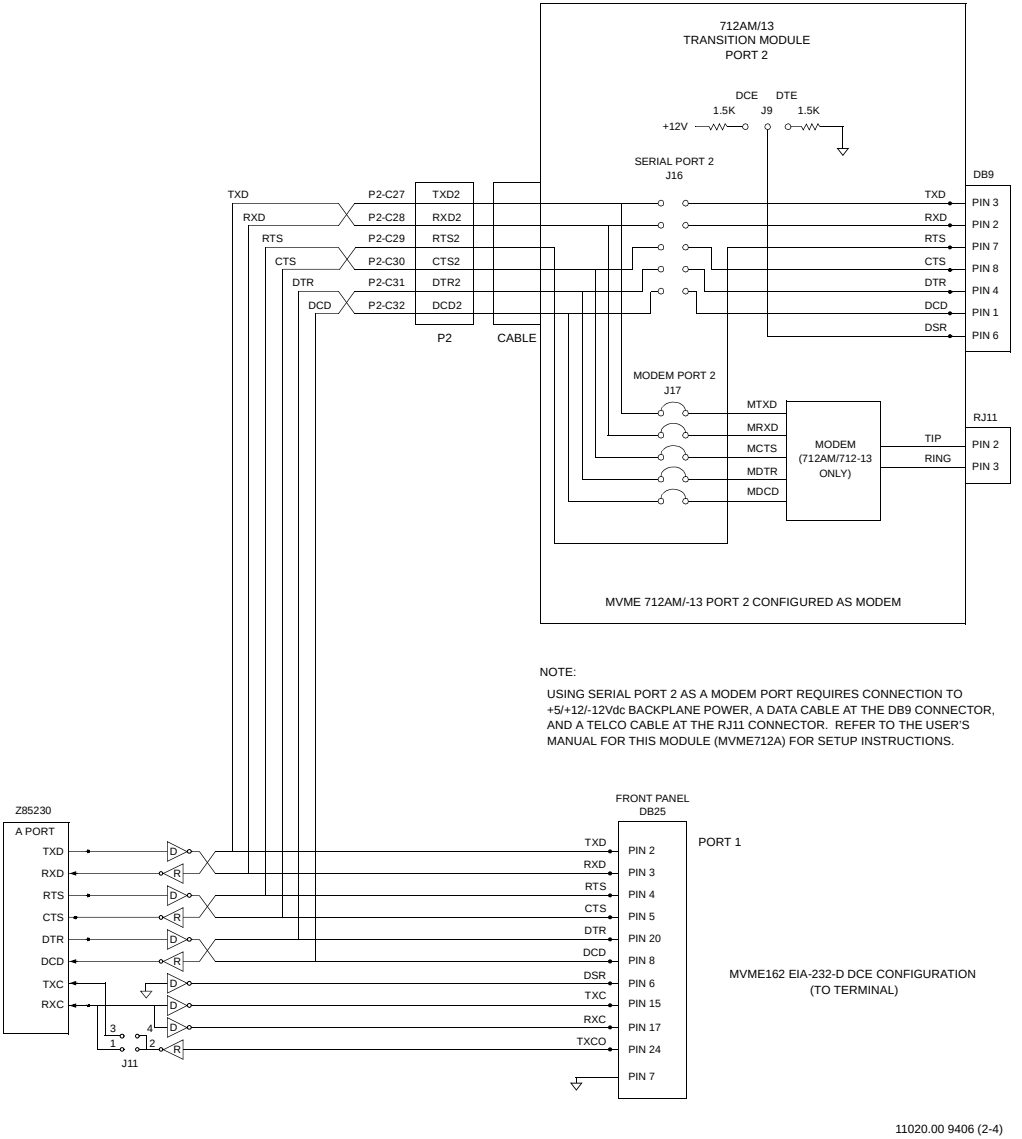
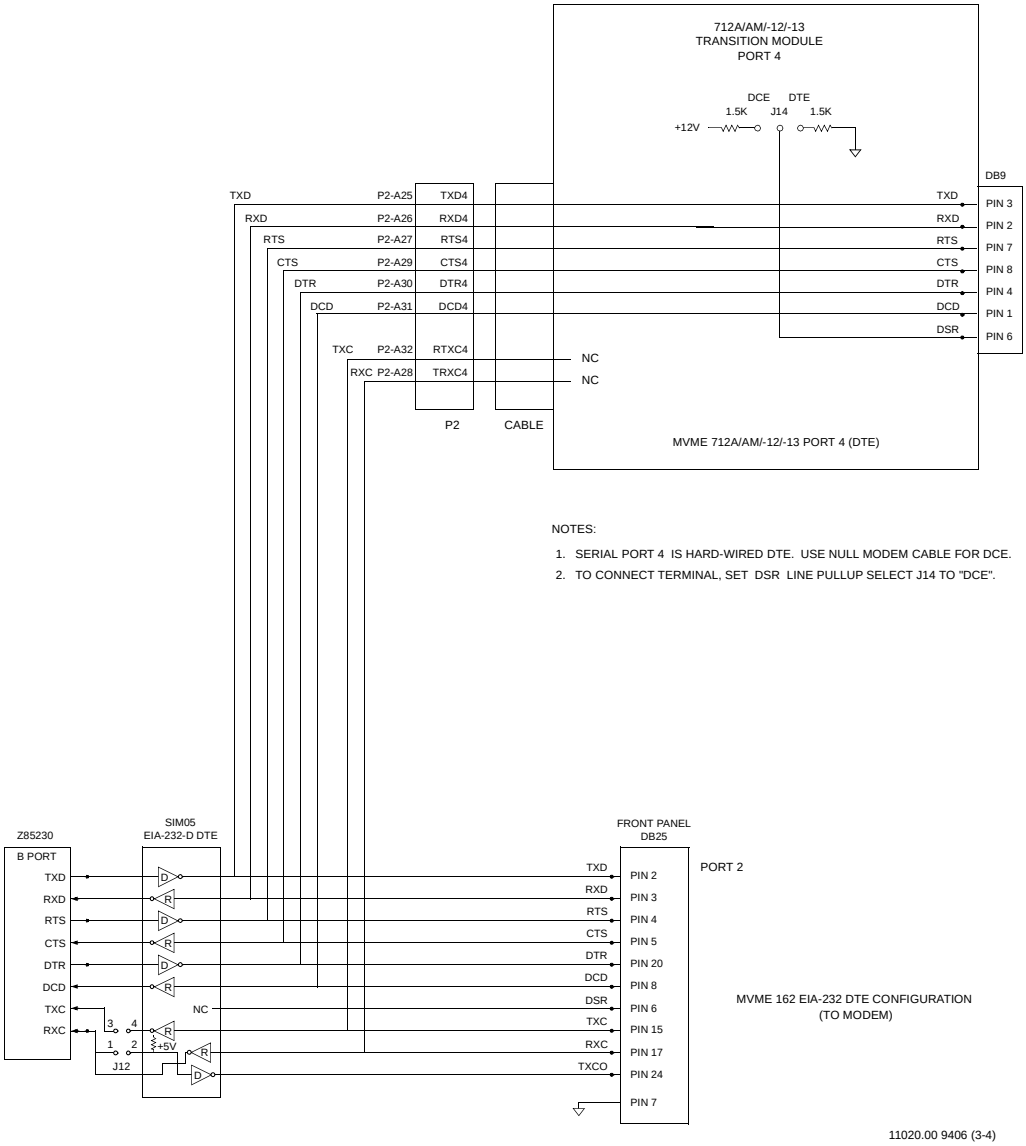


Figure 2-5. MVME162 EIA-232-D Connection Diagram, MVME712A/AM/-12/-13

(Sheet 1 of 4)


712A/AM/-12/-13
TRANSITION MODULE
PORT 4
DCE DTE
1.5K J14 1.5K
+12V
DB9
TXD P2-A25 TXD4 TXD PIN 3
RXD P2-A26 RXD4 RXD PIN 2
RTS P2-A27 RTS4 RTS PIN 7
CTS P2-A29 CTS4 CTS PIN 8
DTR P2-A30 DTR4 DTR PIN 4
DCD P2-A31 DCD4 DCD PIN 1
DSR PIN 6
TXC P2-A32 RTXC4 NC
RXC P2-A28 TRXC4 NC
P2
CABLE
MVME 712A/AM/-12/-13 PORT 4 (DTE)
NOTES:
1. SERIAL PORT 4 IS HARD-WIRED DTE. USE NULL MODEM CABLE FOR DCE.
2. TO CONNECT TERMINAL, SET DSR LINE PULLUP SELECT J14 TO "DCE".
Z85230
B PORT
TXD
RXD
RTS
CTS
DTR
DCD
TXC
RXC
SIM05
EIA-232-D DTE
NC
3 4
1 2
+5V
J12
FRONT PANEL
DB25
PORT 2
TXD PIN 2
RXD PIN 3
RTS PIN 4
CTS PIN 5
DTR PIN 20
DCD PIN 8
DSR PIN 6
TXC PIN 15
RXC PIN 17
TXCO PIN 24
PIN 7
MVME 162 EIA-232 DTE CONFIGURATION
(TO MODEM)
11020.00 9406 (3-4)

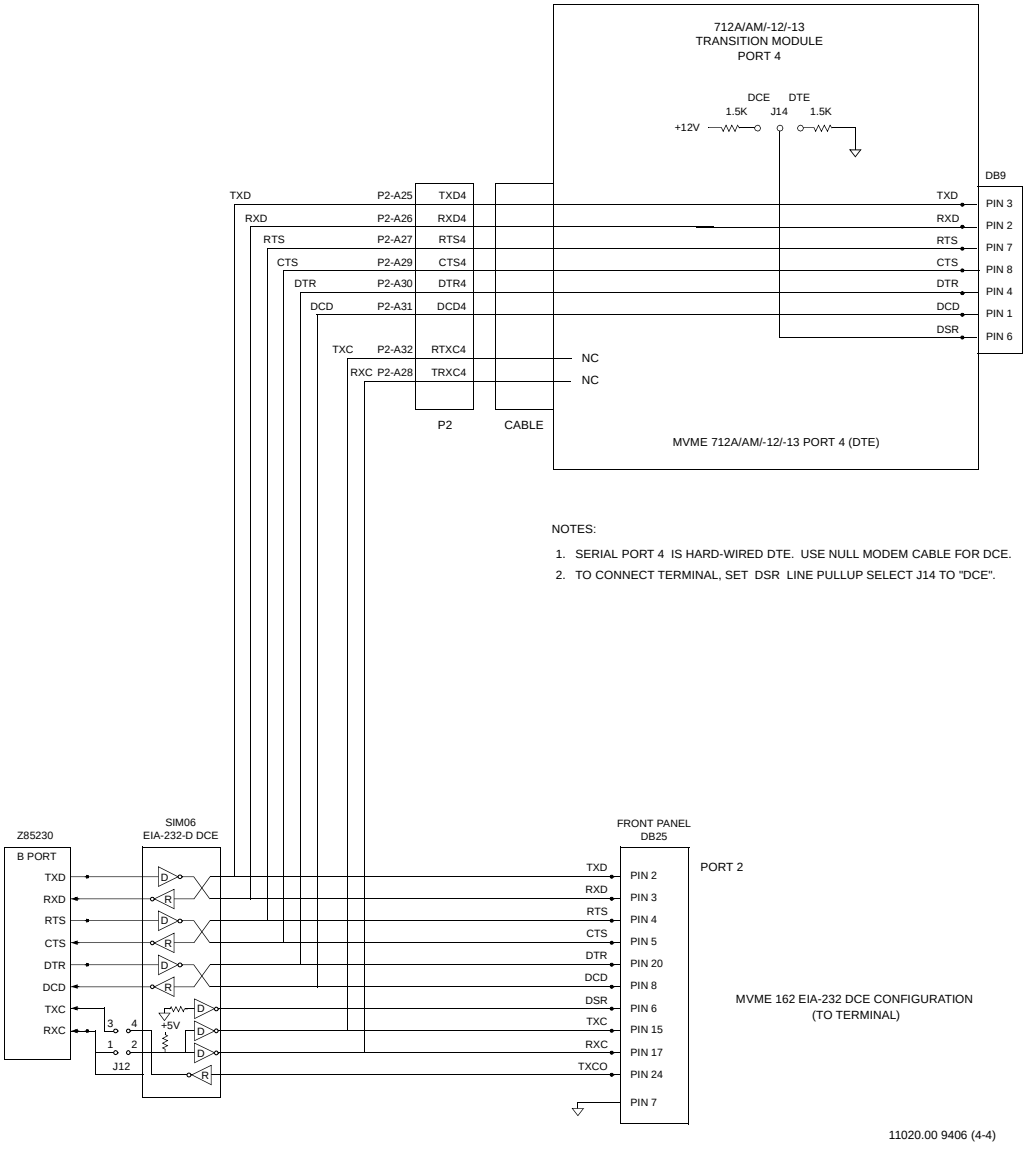


Figure 2-5. MVME162 EIA-23-D Connection Diagram, MVME712A/AM/-12/-13

(Sheet 2 of 4)

Figure 2-5. MVME162 EIA-232-D Connection Diagram, MVME712A/AM/-12/-13

(Sheet 3 of 4)

Figure 2-5. MVME162 EIA-232-D Connection Diagram, MVME712A/AM/-12/-13

(Sheet 4 of 4)

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 MVME162 MC68040- or MC68LC040-based Embedded Controller, and is known as the MVME162BUG, or 162Bug. It includes diagnostics for testing and configuring IndustryPack modules.

Description of 162Bug

The 162Bug package, MVME162Bug, is a powerful evaluation and debugging tool for systems built around the MVME162 CISC-based microcomputers. Facilities are available for loading and executing user programs under complete operator control for system evaluation. 162Bug 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 162Bug routines that handle I/O, data conversion, and string functions are available to user programs through the TRAP #15 system calls.

162Bug consists of three parts:

- A command-driven user-interactive software debugger, described in Chapter 4 and hereafter referred to as "the debugger" or "162Bug".
- A command-driven diagnostic package for the MVME162 hardware, hereafter referred to as "the diagnostics".
- A user interface which accepts commands from the system console terminal.

When using 162Bug, you operate out of either the debugger directory or the diagnostic directory. If you are in the debugger directory, the debugger prompt "`162-Bug>`" is displayed and you have all of the debugger commands at your disposal. If you are in the diagnostic directory, the diagnostic prompt "`162-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 162Bug is command-driven, it performs its various operations in response to user commands entered at the keyboard. When you enter a command, 162Bug 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 162Bug, 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 162Bug 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.

162Bug Implementation

MVME162Bug 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, 162Bug is contained in two of the four 28F020 Flash memories, providing 512KB (128K longwords) of storage. Optionally, the 162Bug can be loaded and executed in a single 27C040 PROM. (128K longwords) of storage. Both memory devices are necessary regardless of how much space is actually occupied by the firmware, because of the 32-bit longword-oriented MC68040 memory bus architecture. The executable code is checksummed at every power-on or reset firmware entry, and the result (which includes a pre-calculated checksum contained in the memory devices), is tested for an expected zero. Thus, users are cautioned against modification of the memory devices unless re-checksum precautions are taken.

Installation and Startup

Even though 162Bug is installed in the Flash memories on the MVME162 module, for 162Bug to operate properly with the MVME162, you must follow the steps below:

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 J22 affect 162Bug 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.

   These readable jumpers can be read as a register (at $FFF4202D) on the Memory Controller (MCchip) ASIC. 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 J22) contains eight bits. Refer also to the *MVME162 Embedded Controller Programmer's Reference Guide* for more information on the MCchip.

   The MVME162Bug reserves / defines the four lower order bits (GPI3 to GPI0). The following is the description for the bits reserved / defined by the debugger:

Bit	J22 Pins	Description
Bit #0 (GPI0)	15-16	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)	13-14	When this bit is a one (high), it instructs the debugger to use the default setup / operation parameters in Flash or PROM 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 Flash / PROM defaults.
Bit #2 (GPI2)	11-12	Reserved for future use.
Bit #3 (GPI3)	9-10	When this bit is a zero (low), it informs the debugger that it is executing out of the Flash memories. When this bit is a one (high), it informs the debugger that it is executing out of the PROM.
Bit #4 (GPI4)	7-8	Open to your application.
Bit #5 (GPI5)	5-6	Open to your application.
Bit #6 (GPI6)	3-4	Open to your application.
Bit #7 (GPI7)	1-2	Open to your application.

Note that when the MVME162 comes up in a cold reset, 162Bug runs in Board Mode. Using the Environment (**ENV**) or **MENU** commands can make 162Bug run in System Mode. Refer to Appendix A.

2. Configure header J1 by installing / removing a jumper between pins 1 and 2. A jumper installed / removed enables / disables the system controller function of the MVME162.
3. You may configure Port B of the Z85230 serial communications controller via a serial interface module (SIM) which is installed at connector J10 on the MVME162 board. Four serial interface modules are available:
   - EIA-232-D DTE (SIM05)
   - EIA-232-D DCE (SIM06)
   - EIA-530 DTE (SIM07)
   - EIA-530 DCE (SIM08)

   For information on removing and / or installing a SIM, refer Chapter 2.

4. Jumpers on headers J11 and J12 configure serial ports 1 and 2 to drive or receive clock signals provided by the TXC and RXC signal lines. The factory configures the module for asynchronous communication, that is, installs no jumpers. Refer to Chapter 2 if your application requires configuring ports 1 and 2 for synchronous communication.
5. If using a PROM version of the 162Bug, install the PROM device in socket U47. Be sure that the physical chip orientation is correct, that is, with the flatted corner of the PROM aligned with the corresponding portion of the PROM socket on the MVME162 module.

Check the jumper installation on header J21 for correct size. Connect pins 1 and 2 on J21 for 27C080 devices, or pins 2 and 3 for 27C040 devices. The factory default is 2 and 3.

Remove the jumper on J22 pins 9 and 10.

6. Refer to the set-up procedure for your particular chassis or system for details concerning the installation of the MVME162.
7. Connect the terminal that is to be used as the 162Bug system console to the default debug EIA-232-D port at serial port 1 on the front panel of the MVME162 module. Refer to Chapter 2 for other connection options. 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 MVME162 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 162Bug debugger.

Note **In order for high-baud rate serial communication between 162Bug and the terminal to work, the terminal must do some form of handshaking. If the terminal being used does not do hardware handshaking via the CTS line, then it must do XON/XOFF handshaking. If you get garbled messages and missing characters, then you should check the terminal to make sure XON/XOFF handshaking is enabled.**

8. If you want to connect devices (such as a host computer system and/or a serial printer) to the other EIA-232-D port connectors (marked SERIAL PORTS 2, 3, and 4 on the MVME712X transition module), connect the

appropriate cables and configure the port(s) as detailed in step 6. above. After power-up, this(these) port(s) can be reconfigured by programming the MVME162 Z85230 Serial Communications Controller (SCC), or by using the 162Bug **PF** command.

9. Power up the system. 162Bug executes some self-checks and displays the debugger prompt "`162-Bug>`" (if 162Bug is in Board Mode). However, if the **ENV** command (Appendix A) has put 162Bug 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.

Autoboot

Autoboot is a software routine that is contained in the 162Bug Flash / PROM 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 MVME162. 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 MVME162 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

As shipped from the factory, 162Bug occupies the first half of the Flash memory. This leaves the second half of the Flash memory and the PROM socket (U47) available for your use. The 162Bug is also available in PROM if your application requires all of the Flash memory. Contact your Motorola sales office for assistance. 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 memory devices (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 MVME162 Flash / PROM 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 162Bug Flash / PROM 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 MVME162 front panel RESET switch initiates a system reset. COLD and WARM reset modes are available. By default, 162Bug is in COLD mode. During COLD reset, a total system initialization takes place, as if the MVME162 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 two serial ports are reconfigured to their default state.

During WARM reset, the 162Bug 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 162Bug environment is ever lost (vector table is destroyed, stack corrupted, etc.).

Abort

Abort is invoked by pressing and releasing the ABORT switch on the MVME162 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 162Bug is approximately 512KB of code, consisting of download, debugger, and diagnostic packages and contained entirely in Flash or PROM.

The 162Bug executes from $FF800000 whether in Flash or PROM. With jumper at J22 pins 9-10 installed (factory ship configuration), the Flash memories appear at address $FF800000 and are the parts executed during reset. With this configuration, the PROM socket is mapped to address $FFA00000. If you remove the jumper at J22 pins 9 and 10, the address spaces of the Flash and PROM are swapped.

The 162Bug initial stack completely changes all 8KB of memory at addresses $FFE0C000 through $FFE0DFFF at power up or reset.

Type of Memory Present	Default DRAM Base Address	Default SRAM Base Address
A single DRAM mezzanine	$00000000	FFE00000 (onboard SRAM)
A single SRAM mezzanine	N / A	$00000000
A DRAM mezzanine stacked with an SRAM mezzanine	$00000000	$E1000000
Two DRAM mezzanines stacked	$00000000	$FFE00000 (onboard SRAM)

DRAM can be ECC or parity type. DRAM mezzanines are mapped in contiguously starting at zero ($00000000), largest first. With two mezzanines of the same size, ECC type DRAM is first. If both are ECC type, the bottom one is first.

The 162Bug requires 2KB of NVRAM for storage of board configuration, communication, and booting parameters. This storage area begins at $FFFC16F8 and ends at $FFFC1EF7.

162Bug 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 162Bug stack and static variable space and the rest is reserved as user space. Whenever the MVME162 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.

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 162Bug 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 162Bug, 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

162Bug can initiate disk input/output by communicating with intelligent disk controller modules over the VMEbus. Disk support facilities built into 162Bug 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 162Bug. 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 162Bug 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. 162Bug 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 162Bug Commands

These following 162Bug 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 162Bug 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 162Bug. It is used as a debugging tool.

Disk I/O via 162Bug System Calls

All operations that actually access the disk are done directly or indirectly by 162Bug 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, 162Bug 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 162Bug Controller and Device Parameters

162Bug initializes the parameter tables for a default configuration of controllers and devices (refer to Appendix B). If the system needs to be configured differently than this default configuration (for example, to use a 70MB Winchester drive where the default is a 40MB Winchester drive), then these tables must be changed.

There are three ways to change the parameter tables:

- Using **BO** or **BH**. When you invoke one of these commands, the configuration area of the disk is read and the parameters corresponding to that device are rewritten according to the parameter information contained in the configuration area. This is a temporary change. If a cold-start reset occurs, then the default parameter information is written back into the tables.
- Using the **IOT**. You can use this command to reconfigure the parameter table manually for any controller and/or device that is different from the default. This is also a temporary change and is overwritten if a cold-start reset occurs.
- Obtain the source. You can then change the configuration files and rebuild 162Bug so that it has different defaults. Changes made to the defaults are permanent until changed again.

Disk I/O Error Codes

162Bug 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 Flash/PROM 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

162Bug returns an error code if an attempted network operation is unsuccessful.

Multiprocessor Support

The MVME162 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 MVME162 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:

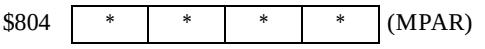
HEX	0	(HEX 00)	--	Wait. Initialization not yet complete.
ASCII	E	(HEX 45)	-	Code pointed to by the MPAR address is executing.
ASCII	P	(HEX 50)	-	Program Flash Memory. The MPAR is set to the address of the Flash memory program control packet.
ASCII	R	(HEX 52)	-	Ready. The firmware monitor is watching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

You can only program Flash memory by the MPCR method. Refer to the .PFLASH system call in the *MVME162Bug Debugging Package 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 MVME162 dual-port RAM by issuing a remote **GO** command using the VMEchip2 Global Control and Status Registers (GCSR). The remote processor places the MVME162 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 MVME162 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 MVME162. 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 162Bug package includes a set of hardware diagnostics for testing and troubleshooting the MVME162. To use the diagnostics, switch directories to the diagnostic directory. If you are in the debugger directory, you can switch to the diagnostic directory with the debugger command Switch Directories (**SD**). The diagnostic prompt ("`162-Diag>`") appears. Refer to the *MVME162Bug 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.

Manufacturing Test Process

During the manufacturing process for MVME162 modules, the manufacturing test parameters and testing state flags are stored in NVRAM. These strings are installed during the manufacturing process and result in the product performing manufacturing tests. None of these tests harm the product or system into which a module is installed. Entering an ASCII break on the console port from a terminal terminates these tests.

The two state flags that start the test processes are:

```
FLASH EMPTY$00122984
```

and

```
Burnin test$00000000
```

If either string is in the first location of NVRAM ($FFFC0000), the test process starts.

This note is to inform users about the manufacturing test process; it is not intended to instruct customers in its use. Motorola reserves the right to delete, change, or modify this process.

USING THE 162Bug DEBUGGER

4

Entering Debugger Command Lines

162Bug is command-driven and performs its various operations in response to user commands entered at the keyboard. When the debugger prompt (`162-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 162Bug 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:

```
162Bug>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
162Bug>
```

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

```
162Bug>OF R0
R0 =00000000 00000000? 1327C. <CR>
162Bug>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
162Bug>
```

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

Port Numbers

Some 162Bug 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. MVME162 EIA-232-D Debug (Terminal Port 0 or 00) (PORT 1 on the MVME162 P2 connector). Sometimes known as the "console port", it is used for interactive user input/output by default.
2. MVME162 EIA-232-D (Terminal Port 1 or 01) (PORT 2 on the MVME162 P2 connector). Sometimes known as the "host port", this is the default for downloading, uploading, concurrent mode, and transparent modes.

Note **These logical port numbers (0 and 1) are shown in the pinouts of the MVME162 module as "SERIAL PORT 1" and "SERIAL PORT 2", respectively. Physically, they are all part of connector P2. They are also available at the front panel DB-25 connectors J15 (for PORT 1 or A) and J9 (for PORT 2 or B).**

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 162Bug 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 162Bug **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 MVME162 port 1. (Hardware configuration details are in the section on *Installation and Startup* in Chapter 3.) The file is downloaded from the host to MVME162 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.

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 162Bug 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. 162Bug uses certain of the MVME162 onboard resources and also offboard system memory to contain temporary variables, exception vectors, etc. If you disturb resources upon which 162Bug 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.

162Bug Vector Table and Workspace

As described in the *Memory Requirements* section in Chapter 3, 162Bug needs 64KB of read / write memory to operate. The 162Bug 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, 162Bug reserves space for static variables and initializes these static variables to predefined default values. After the static variables, 162Bug 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 162Bug

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 162Bug

Vector Offset	Exception	162Bug 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 1	Level 7 interrupt	ABORT pushbutton
$NOTE 2	Level 7 interrupt	AC Fail
$DC	FP Unimplemented Data Type	Software emulation and data type conversion of floating point data.

NOTES: 1. This depends on what the Vector Base Register (VBR) is set to in the MCchip.

2. This depends 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.

```
162Bug>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
162Bug>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
162Bug>
```

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 162Bug or it may create a separate exception vector table of its own. The two following sections detail these two methods.

Using 162Bug Target Vector Table

The 162Bug 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 162Bug 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 *162Bug 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 162Bug 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 162Bug vector table into the corresponding vector locations in your program vector table.

The vector for the 162Bug generalized exception handler (described in detail in the *162Bug 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 162Bug target program vector table (which your program saved), yielding the address of the 162Bug exception vector. The program then jumps to the address stored at this vector location, which is the address of the 162Bug 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.
```


162Bug Generalized Exception Handler

The 162Bug 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.

```
162Bug>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
162Bug>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
162Bug>
```

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.

```
162Bug>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             '...........
162Bug>
```

Floating Point Support

The floating point unit (FPU) of the MC68040 microprocessor chip is supported in 162Bug. For MVME162Bug, 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 162Bug Debugger Command Set

The 162Bug 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 *MVME162Bug 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 *MVME162 Embedded Controller User's Manual* for the actual location. The information block contains various elements detailing specific operation parameters of the hardware. The *MVME162 Embedded Controller 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.

Although the factory fills all fields except the IndustryPack fields, only these fields MUST contain correct information:

- ❑ MPU clock speed
- ❑ Ethernet address
- ❑ Local SCSI identifier

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

```
162-Bug>cnfg
Board (PWA) Serial Number = "000000061050"
Board Identifier          = "MVME162-03      "
Artwork (PWA) Identifier  = "01-W3814B03A    "
MPU Clock Speed           = "2500"
Ethernet Address          = 08003E20A867
Local SCSI Identifier     = "07"
Parity Memory Mezzanine Artwork (PWA) Identifier  = "        "
Parity Memory Mezzanine (PWA) Serial Number       = "        "
Static Memory Mezzanine Artwork (PWA) Identifier  = "        "
Static Memory Mezzanine (PWA) Serial Number       = "        "
ECC Memory Mezzanine #1 Artwork (PWA) Identifier  = "        "
ECC Memory Mezzanine #1 (PWA) Serial Number       = "        "
ECC Memory Mezzanine #2 Artwork (PWA) Identifier  = "        "
ECC Memory Mezzanine #2 (PWA) Serial Number       = "        "
```

```
Serial Port 2 Personality Artwork (PWA) Identifier   = "       "
Serial Port 2 Personality Module (PWA) Serial Number = "       "
IndustryPack A Board Identifier         = "        "
IndustryPack A (PWA) Serial Number      = "        "
IndustryPack A Artwork (PWA) Identifier = "        "
IndustryPack B Board Identifier         = "        "
IndustryPack B (PWA) Serial Number      = "        "
IndustryPack B Artwork (PWA) Identifier = "        "
IndustryPack C Board Identifier         = "        "
IndustryPack C (PWA) Serial Number      = "        "
IndustryPack C Artwork (PWA) Identifier = "        "
IndustryPack D Board Identifier         = "        "
IndustryPack D (PWA) Serial Number      = "        "
IndustryPack D Artwork (PWA) Identifier = "        "
162-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]	B	Bug mode
Field Service Menu Enable [Y/N]	N	Do not 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 the appropriate system busses (e.g., VMEbus, local bus) 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
Industry Pack Reset on Debugger Startup [Y/N]	N	Industry Pack(s) is/are not reset on debugger startup.
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.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Auto Boot Abort Delay	15	This is the time in seconds that the Auto Boot sequence will delay before starting the boot. The purpose for the delay is to allow you the option of stopping the boot by use of the Break key. The time value is from 0 through 255 seconds.
Auto Boot Default String [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	FFDFFFFC	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.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
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.
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-162 environment, each MVME162 board could be set to start its work page at a unique address to allow multiple debuggers to operate simultaneously.
Memory Search Ending Address	00100000	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 MVME162. Default Memory Search Ending Address is the calculated size of local memory.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Memory Search Increment Size	00010000	This multi-CPU feature is used to offset the location of the Bug work page. This must be a multiple of the debugger work page, modulo $10000 (64KB). Typically, Memory Search Increment Size is the product of CPU number and size of the Bug work page. Example: first CPU $0 (0 x $10000), second CPU $10000 (1 x $10000), etc.
Memory Search Delay Enable [Y / N]	N	There will be no delay before the Bug begins its search for a work page.
Memory Search Delay Address	FFFFD20F	Default address is $FFFFD20F. This is the MVME162 GCSR GPCSR0 as accessed through VMEbus A16 space and assumes the MVME162 GRPAD (group address) and BDAD (board address within group) switches are set to "on". This byte-wide value is initialized to $FF by MVME162 hardware after a System or Power-on Reset. In a multi-162 environment, where the work pages of several Bugs are to reside in the memory of the primary (first) MVME162, 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	00100000	Default Ending Address is the calculated size of local memory.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Base Address of Dynamic Memory	00000000	Beginning address of Dynamic Memory (Parity and/or ECC type memory). It must be a multiple of the Dynamic Memory board size, starting with 0. Default is $0.
Size of Parity Memory	00100000	This is the size of the Parity type dynamic RAM mezzanine, if any. The default is the calculated size of the Dynamic memory mezzanine board.
Size of ECC Memory Board #0	00000000	This is the size of the first ECC type memory mezzanine. The default is the calculated size of the memory mezzanine.
Size of ECC Memory Board #1	00000000	This is the size of the second ECC type memory mezzanine. The default is the calculated size of the memory mezzanine.
Base Address of Static Memory	FFE00000	This is the beginning address of SRAM. The default for this parameter is FFE00000 for the onboard 512KB, or E1000000 for the 2MB SRAM mezzanine. If only 2 MB SRAM is present, it defaults to address 00000000.
Sizeof Static Memory	00080000	This is the size of the SRAM type memory present. The default is the calculated size of the onboard SRAM or an SRAM type mezzanine.
ENV asks the following series of questions to set up the VMEbus interface for the MVME162 series modules. You should have a working knowledge of the VMEchip2 as given in the *MVME162* Embedded Controller Programmer's Reference Guide in order to perform this configuration. Also included in this series are questions for setting ROM and Flash access time. The slave address decoders are used to allow another VMEbus master to access a local resource of the MVME162. There are two slave address decoders set. They are set up as follows:		
Slave Enable #1 [Y/N]	Y	Yes, set up and enable the Slave Address Decoder #1.
Slave Starting Address #1	00000000	Base address of the local resource that is accessible by the VMEbus. Default is the base of local memory, $0.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Slave Ending Address #1	000FFFFF	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]	N	Do not set up and enable the Slave Address Decoder #2.
Slave Starting Address #2	00000000	Base address of the local resource that is accessible by the VMEbus. Default is 0.
Slave Ending Address #2	00000000	Ending address of the local resource that is accessible by the VMEbus. Default is 0.
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	0000	Defines the access restriction for the address space defined with this slave address decoder. Default is $0000.
Master Enable #1 [Y / N]	Y	Yes, set up and enable the Master Address Decoder #1.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
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 set up and enable the Master Address Decoder #2.
Master Starting Address #2	00000000	Base address of the VMEbus resource that is accessible from the local bus. Default is $00000000.
Master Ending Address #2	00000000	Ending address of the VMEbus resource that is accessible from the local bus. Default is $00000000.
Master Control #2	00	Defines the access characteristics for the address space defined with this master address decoder. Default is $00.
Master Enable #3 [Y / N]	Y	Yes, set up 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.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
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 more 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.
Master Address Translation Address #4	00000000	This register will allow the VMEbus address and the local address to be different. The value in this register is the base address of VMEbus resource that is associated with the starting and ending address selection from the previous questions. Default is 0.
Master Address Translation Select #4	00000000	This register defines which bits of the address are significant. A logical one "1" indicates significant address bits, logical zero "0" is non-significant. Default is 0.
Master Control #4	00	Defines the access characteristics for the address space defined with this master address decoder. Default is $00.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
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 Access Time Code	03	This defines the ROM access time. The default is $03, which sets an access time of 180 ns.
Flash Access Time Code	02	This defines the FLASH access time. The default is $02, which sets an access time of 140 ns.
MCC Vector Base VMEC2 Vector Base #1 VMEC2 Vector Base #2	05 06 07	Base interrupt vector for the component specified. Default: MCchip = $05, VMEchip2 Vector 1 = $06, VMEchip2 Vector 2 = $07.
VMEC2 GCSR Group Base Address	D2	Specifies the group address ($FFFFXX00) in Short I/O for this board. Default = $D2.
VMEC2 GCSR Board Base Address	00	Specifies the base address ($FFFFD2XX) 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	02	This controls the local bus timeout. Default $02 = 256 µs.
VMEbus Access Time Out Code	02	This controls the local bus to VMEbus access timeout. Default $02 = 32 ms.

Configuring the IndustryPacks

ENV asks the following series of questions to set up IndustryPacks (IP) on MVME162 modules.

The *MVME162 Embedded Controller Programmer's Reference Guide* describes the base addresses and the IP register settings. Refer to that manual for information on setting base addresses and register bits.

```
IP A Base Address              = 00000000?
IP B Base Address              = 00000000?
IP C Base Address              = 00000000?
IP D Base Address              = 00000000?
```

Base address for mapping IP modules. Only the upper 16 bits are significant.

```
IP D/C/B/A Memory Size         = 00000000?
```

Define the memory size requirements for the IP modules:

Bits	IP	Register Address
31-24	D	FFFBC00F
23-16	C	FFFBC00E
15-08	B	FFFBC00D
07-00	A	FFFBC00C

```
IP D/C/B/A General Control     = 00000000?
```

Define the general control requirements for the IP modules:

Bits	IP	Register Address
31-24	D	FFFBC01B
23-16	C	FFFBC01A
15-08	B	FFFBC019
07-00	A	FFFBC018

```
IP D/C/B/A Interrupt 0 Control = 00000000?
```

Define the interrupt control requirements for the IP modules channel 0:

Bits	IP	Register Address
31-24	D	FFFBC016
23-16	C	FFFBC014
15-08	B	FFFBC012
07-00	A	FFFBC010

```
IP D/C/B/A Interrupt 1 Control = 00000000?
```

Define the interrupt control requirements for the IP modules channel 1:

Bits	IP	Register Address
31-24	D	FFFBC017
23-16	C	FFFBC015
15-08	B	FFFBC013
07-00	A	FFFBC011

DISK/TAPE CONTROLLER DATA

B

Disk/Tape Controller Modules Supported

The following VMEbus disk / tape controller modules are supported by the 162Bug. 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 Embedded Controller	$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 an MVME162 with a SCSI port is used, then the MVME162 module has CLUN 0.

(2) For MVME162s, 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 Embedded Controllers -- 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 and MVME162.

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 MVME162Bug. The default address for each type and position is showed to indicate where the controller must reside to be supported by the MVME162Bug. 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
MVME162	$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.

C

D

INDEX

E

F

G

H

I

INDEX

J

L

M

N

O

P

T

U

V

W

X

Z

INDEX