Motorola MVME172-533

VME Embedded Controller

In Stock

Used and in Excellent Condition

Open Web Page

https://www.artisantg.com/98630-2

MVME172
VME Embedded Controller
Installation and Use

VME172A/IH1

Notice

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

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

Restricted Rights Legend

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

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

Motorola, Inc.
Computer Group
2900 South Diablo Way
Tempe, Arizona 85282-9602

Preface

This document provides general information and basic installation instructions for the MVME172 VME Embedded Controller (which is available in the versions listed below).

Assembly Item	Board Description
MVME172-213	64MHz, 4MB DRAM
MVME172-223	60MHz, 4MB DRAM
MVME172-233	64MHz, 4MB ECC DRAM
MVME172-243	60MHz, 4MB ECC DRAM
MVME172-253	64MHz, 16MB ECC DRAM
MVME172-263	60MHz, 16MB ECC DRAM

Assembly Item	Board Description
MVME172-303	60MHz, 8MB DRAM
MVME172-313	64MHz, 8MB DRAM
MVME172-323	60MHz, 8MB ECC DRAM
MVME172-333	64MHz, 8MB ECC DRAM
MVME172-343	60MHz, 32MB ECC DRAM
MVME172-353	64MHz, 32MB ECC DRAM
MVME172-363	60MHz, 16MB DRAM
MVME172-373	60MHz, 16MB DRAM

The MVME172 Installation Guide provides general board level hardware description, hardware preparation and installation instructions, debugger general information, and using the debugger; for the MVME172 VME Embedded Controller.

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

A basic knowledge of computers and digital logic is assumed.

Companion publications are listed beginning on page 1-3.

Safety Summary
Safety Depends On You

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

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

Ground the Instrument.

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

Do Not Operate in an Explosive Atmosphere.

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

Keep Away From Live Circuits.

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

Do Not Service or Adjust Alone.

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

Use Caution When Exposing or Handling the CRT.

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

Do Not Substitute Parts or Modify Equipment.

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

Dangerous Procedure Warnings.

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

WARNING

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

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

WARNING

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

CE

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

EN55022 (CISPR 22) Radio Frequency Interference

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

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

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

Printed in the United States of America
March 1997

Contents

List of Figures

List of Tables

Board Level Hardware Description

1

Introduction

This chapter describes the board level hardware features of the MVME172 VME 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 re included in the detailed hardware functional description. The chapter closes with some general memory maps.

All programmable registers in the MVME172 that reside in ASICs are covered in the *MVME172 Embedded Controller Programmer's Reference Guide*.

Overview

The MVME172 is based on the MC68060/MC68LC060 microprocessor. Various versions of the MVME172 have 4, 8, or 16MB of parity-protected DRAM or 4, 8, 16, 32, or 64MB of ECC-protected DRAM; 128KB of SRAM (with battery backup); time-of-day clock (with battery backup); an optional LAN Ethernet transceiver interface; four serial ports with EIA-232-D interface; six tick timers with watchdog timer(s); two EPROM sockets; 2MB Flash memory (one Flash device); two IndustryPack (IP) interfaces with DMA; optional SCSI bus interface with DMA; and an optional VMEbus interface (local bus to VMEbus/VMEbus to local bus, with A16/A24/A32, D8/D16/D32 bus widths and a VMEbus system controller).

Input/Output (I/O) signals are routed through industry-standard connectors on the MVME172 front panel; no adapter boards or transition modules are required. I/O connections include an optional 68-pin SCSI connector, an optional DB-15 Ethernet connector, and four 8-pin RJ-45 serial connectors on the front panel. In addition, the panel has cutouts for routing of flat cables to the optional IndustryPack modules.

The following ASICS are used on the MVME172:

- **VMEchip2.** (VMEbus interface). Provides 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 are D8, D16, or D32. VMEchip2 DMA transfers to the VMEbus, however, are D16, D32, D16/BLT, D32/BLT, or D64/MBLT.

- **MC2chip.** Provides four tick timers, the interface to the LAN chip, SCSI chip, serial port chip, BBRAM, EPROM/Flash, parity-DRAM and SRAM.

- **MCECC memory controller.** Provides the programmable interface for the ECC-protected 16MB DRAM mezzanine board.

- **IndustryPack Interface Controller (IP2).** The IP2 provides control and status information for up to two singlewide IPs or one doublewide IP that can be plugged into the MVME172 main board.

Related Documentation

The MVME172 ships with an installation and use manual (the document you are presently reading, Motorola Publications Number VME172A/IH) which includes installation instructions, jumper configuration information, memory maps, debugger/monitor commands, and any other information needed to startup the board.

If you wish to develop your own applications or need more detailed information about your MVME172 VME Embedded Controller, you may purchase the additional documentation listed on the following pages through your local Motorola sales office.

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

Document Set for MVME172

When they are available, you may purchase the following manuals (individually or as a set) from your local Motorola sales office. The document set LK-172SET includes:

Motorola Publication Number	Description
VME172A/PG	MVME172 VME Embedded Controller Programmer's Reference Guide
V172DIAA/UM	MVME172Bug Debugging Package User's Manual
68KBUG1/D 68KBUG2/D	Debugging Package for Motorola 68K CISC CPUs User's Manual (Parts 1 and 2)
SBCSCSI/D	Single Board Computers SCSI Software User's Manual

Other Applicable Motorola Publications

The following publications are applicable to the MVME172 and may provide additional helpful information. They may be purchased through your local Motorola sales office.

Motorola Publication Number	Description
M68000FR	M68000 Family Reference Manual
M68060UM	MC68060 Microprocessors User's Manual
M68040UM	MC68040 Microprocessors User's Manual

Non-Motorola Peripheral Controllers Publications Bundle

For your convenience, we have collected user's manuals for each of the peripheral controllers used on the MVME172 from the suppliers. This bundle (part number **68-1X7DS**) may be purchased through your local Motorola sales office. It includes the following manuals:

Part Number	Description
NCR53C710DM	NCR 53C710 SCSI I/O Processor Data Manual
NCR53C710PG	NCR 53C710 SCSI I/O Processor Programmer's Guide
CL-CD2400/2401	Cirrus Logic CD2401 Serial Controller User's Manual
UM95SCC0100	Zilog Z85230 Serial Communications Controller User's Manual
290218	Intel Networking Components Data Manual
290435	Intel i28F008 Flash Memory Data Sheet
290245	Intel i28F020 Flash Memory Data Sheet
292095	Intel i28F008SA Software Drivers Application Note

Part Number	Description
292099	Intel i28F008SA Automation and Algorithms Application Note
MK48T08/18B	SGS-THOMSON MK48T08 Time Clock/NVRAM Data Sheet
MC68230/D	MC68230 Parallel Interface Timer (PI/T) Data Sheet
SBCCOMPS/L	Customer Letter for Component Alternatives

Applicable Non-Motorola Publications

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

Document Title	Source
VME64 Specification, order number ANSI/VITA 1-1994 **Note:** An earlier version of the VME specification is available as: Versatile Backplane Bus: VMEbus, ANSI/IEEE Std 1014-1987 (VMEbus Specification) (This is also Microprocessor System Bus for 1 to 4 Byte Data, IEC 821 BUS)	VITA (VMEbus International Trade Association) 7825 E. Gelding Dr., Ste. 104 Scottsdale, AZ 85260-3415
ANSI Small Computer System Interface-2 (SCSI-2), Draft Document X3.131-198X, Revision 10c	Global Engineering Documents 15 Inverness Way East Englewood, CO 80112-5704
82596CA Local Area Network Coprocessor Data Sheet, order number 290218; and 82596 User's Manual, order number 296853	Intel Corporation Literature Sales P.O. Box 58130 Santa Clara, CA 95052-8130
28F016SA Flash Memory Data Sheet, order number 290435	Intel Corporation Literature Sales P.O. Box 7641, Mt. Prospect, IL 60056-7641

Document Title	Source
NCR 53C710 SCSI I/O Processor Data Manual, order number NCR53C710DM	NCR Corporation Microelectronics Products Division 1635 Aeroplaza Dr. Colorado Springs, CO 80916
NCR 53C710 SCSI I/O Processor Programmer's Guide, order number NCR53C710PG	
MK48T58(B) Timekeeper ™ and 8Kx8 Zeropower ™ RAM data sheet in Static RAMs Databook, order number DBSRAM71	SGS-THOMSON Microelectronics Group Marketing Headquarters 1000 East Bell Rd. Phoenix, AZ 85022-2699
IndustryPack Logic Interface Specification, Revision 1.0, order number ANSI/VITA 4-1995	VITA (VMEbus International Trade Association) 7825 E. Gelding Dr., Ste. 104 Scottsdale, AZ 85260-3415
Z85230 Serial Communications Controller Data Sheet	Zilog Inc. 210 Hacienda Ave. Campbell, CA 95008-6609

Requirements

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

- VME64 Specification, VITA
- EIA-232-D Serial Interface Specification, EIA
- SCSI Specification, ANSI
- IndustryPack Specification, VITA

Features

- The MC68060 has a clock frequency of 60 MHz; the MC68LC060 has a clock frequency of 64 MHz.
- 4, 8, 16MB of DRAM with parity protection on a mezzanine module, or 4, 8, 16, 32, or 64MB of ECC-protected DRAM
- 128KB of SRAM with battery backup
- Two JEDEC standard 32-pin DIP PROM sockets
- One Intel 28F016SA 2M x 8 Flash memory device with write protection (optional)
- Status LEDs for FAIL, RUN, SCON, and FUSES
- 8K by 8 Non-Volatile RAM (NVRAM) and time-of-day (TOD) clock with battery backup
- RESET and ABORT switches
- Four 32-bit Tick Timers and Watchdog Timer (in the MC2chip ASIC) for periodic interrupts
- Two 32-bit Tick Timers and Watchdog Timer (in the VMEchip2 ASIC) for periodic interrupts
- Eight software interrupts (for MVME172 versions that have the VMEchip2)

- I/O
  - Optional SCSI Bus interface with DMA
  - Four serial ports with EIA-232-D interface (serial port controllers are the Z85230 chips)
  - Optional Ethernet transceiver interface with DMA
  - Two IP interfaces with two channel DMA
- 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)

Specifications

Table 1-1 lists the specifications for an MVME172 without IPs.

Table 1-1. MVME172 Specifications

Characteristics	Specifications
Power requirements (with EPROMs; without IPs)	+5Vdc (± 5%), 3.5 A typical, 4.5 A maximum +12 Vdc (± 5%), 100 mA maximum -12 Vdc (± 5%), 100 mA maximum
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 PC board with mezzanine module only Height Depth Thickness PC board with connectors and front panel Height Depth Thickness	Double-high VMEboard 9.2 inches (233 mm) 6.3 inches (160 mm) 0.66 inch (17 mm) 10.3 inches (262 mm) 7.4 inches (188 mm) 0.80 inch (20 mm)
*Refer to *Cooling Requirements* on page 1-10 and *Special Considerations for Elevated-Temperature Operation* on page 1-11.	

Cooling Requirements

The Motorola MVME172 VME Embedded Controller is specified, designed, and tested to operate reliably with an incoming air temperature range from 0˚ to 55˚ C (32˚ to 131˚ F) with forced air cooling at a velocity typically achievable by using a 100 CFM axial fan. Temperature qualification is performed in a standard Motorola VMEsystem chassis. Twenty-five-watt load boards are inserted in two card slots, one on each side, adjacent to the board under test, to simulate a high power density system configuration. An assembly of three axial fans, rated at 100 CFM per fan, is placed directly under the VME card cage. The incoming air temperature is measured between the fan assembly and the card cage, where the incoming airstream first encounters the controller under test. Test software is executed as the controller is subjected to ambient temperature variations. Case temperatures of critical, high power density integrated circuits are monitored to ensure component vendors specifications are not exceeded.

While the exact amount of airflow required for cooling depends on the ambient air temperature and the type, number, and location of boards and other heat sources, adequate cooling can usually be achieved with 10 CFM and 490 LFM flowing over the controller. Less airflow is required to cool the controller in environments having lower maximum ambients. Under more favorable thermal conditions, it may be possible to operate the controller reliably at higher than 55˚ C with increased airflow. It is important to note that there are several factors, in addition to the rated CFM of the air mover, which determine the actual volume and speed of air flowing over the controller.

Special Considerations for Elevated-Temperature Operation

The following information is for users whose applications for the MVME172 may subject it to high temperatures.

The MVME172 uses commercial-grade devices. Therefore, it can operate in an environment with ambient air temperatures from 0° C to 70° C. Several factors influence the ambient temperature seen by components on the MVME172. Among them are inlet air temperature; air flow characteristics; number, types, and locations of IP modules; power dissipation of adjacent boards in the system, etc.

A temperature profile of the MVME162-223 was developed in an MVME945 12-slot VME chassis. This board was loaded with one GreenSpring IP-Dual P/T module (position a) and one GreenSpring IP-488 module (position b). One 25W load board was installed adjacent to each side of the board under test. The exit air velocity was approximately 200 LFM between the MVME172 and the IP-Dual P/T module. Under these conditions, 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 MVME172 were calculated (from the measured case temperatures) and did not exceed 100° C.

Caution

For elevated-temperature operation, perform similar measurements and calculations to determine the actual operating margin for your specific environment.

To facilitate elevated-temperature operation:

1. Position the MVME172 in the chassis to allow for maximum airflow over the component side of the board.
2. Do not place boards with high power dissipation next to the MVME172.
3. Use low-power IP modules only.

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

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 is read by software to determine operational or exception conditions.

Block Diagram

Refer to Figure 1-1 on page 1-14 for a block diagram of the MVME172.

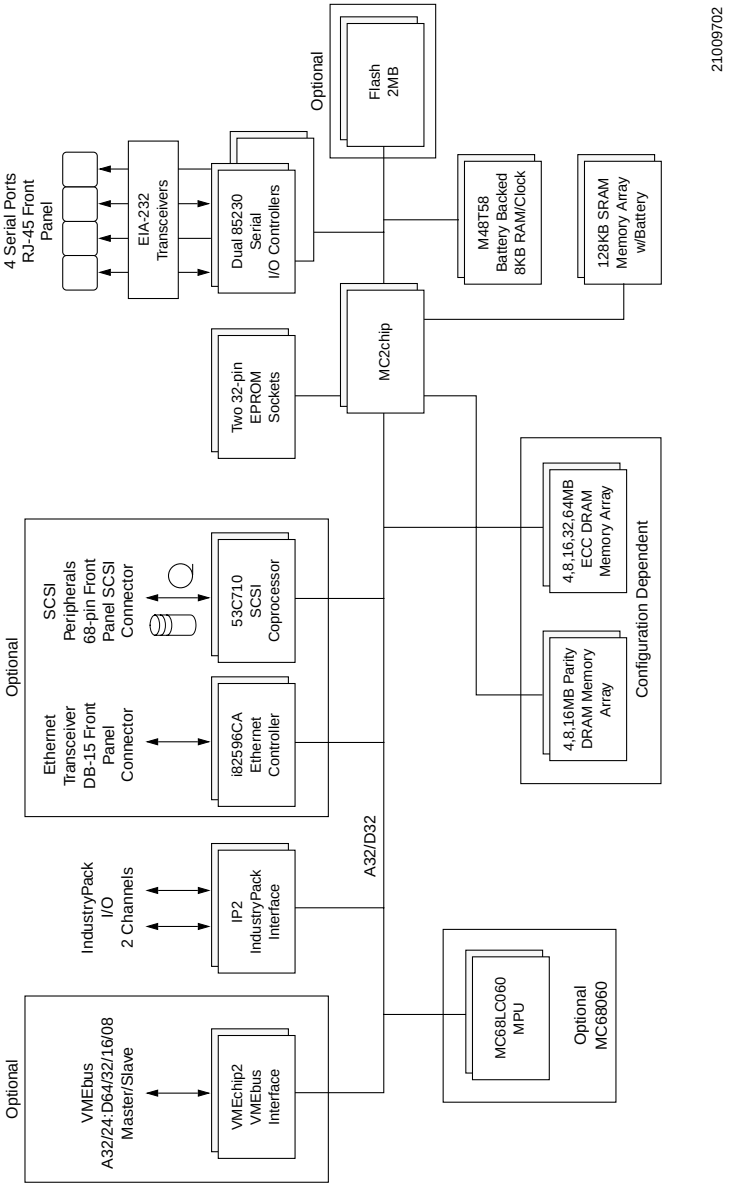


Figure 1-1. MVME172 Block Diagram

Functional Description

This section contains a functional description of the major blocks on the MVME172.

Front Panel Switches and Indicators

There are two switches and four LEDs on the front panel of the MVME172.

- RESET switch. Resets all onboard devices and drives SYSRESET* if the board is system controller. The RESET switch may be disabled by software.
- ABORT switch. 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 172Bug debugger.
- FAIL LED (red). Lights when the BRDFAIL* signal line is active or when the processor is halted. Part of DS1.
- RUN LED (green or amber). Lights when the local bus TIP* signal line is low. This indicates one of the local bus masters is executing a local bus cycle. Part of DS1.
- SCON LED (green). Lights when the VMEchip2 in the MVME172 is the VMEbus system controller. Part of DS2.
- FUSES LED (green). Lights when +5 Vdc, +12 Vdc, and -12 Vdc power is available to the LAN and SCSI interfaces and IP connectors. Part of DS2.

Data Bus Structure

The local bus on the MVME172 is a 32-bit synchronous bus that is based on the MC68060 bus, and supports burst transfers and snooping. The various local bus master and slave devices use the local bus to communicate. The local bus is arbitrated by priority type arbiter and the priority of the local bus masters from highest to lowest is: 82596CA LAN, 53C710 SCSI, VMEbus, and MPU. In the general case, any master can access any slave; however, not all combinations pass the common sense test. Refer to the *MVME172 VME 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.

Microprocessor

The MVME172 may be ordered with an MC68060 or MC68LC060 microprocessor.

The MC68060 has on-chip instruction and data caches and a floating point processor. (The floating point coprocessor is the major difference between the MC68060 and MC68LC060.) Refer to the M68060 user's manual for more information.

MC68*XX*060 Cache

The MVME172 local bus masters (VMEchip2, MC68060/MC68LC060, 53C710 SCSI controller, and 82596CA Ethernet controller) have programmable control of the snoop/caching mode. The MVME172 local bus slaves which support MC68060/MC68LC060 bus snooping are defined in Table 1-3 on page 1-28. The Industry Pack DMA has jumpers to control the state of the snoop control signals.

Note The MC68XX060 has different snoop capabilities than the MC68XX040. Software must take these differences into consideration.

No VMEbus Interface Option

The MVME172 may be operated as an embedded controller without the VMEbus interface. To support this feature, certain logic in the VMEchip2 has been duplicated in the MC2chip. This logic is inhibited in the MC2chip when the VMEchip2 is present. The enables for these functions are controlled by software and MC2chip hardware initialization.

Memory Options

The following memory options are used on the different versions of MVME172 boards.

DRAM Options

The MVME172 offers the following DRAM options: either 4, 8, or 16MB shared DRAM with programmable parity on a mezzanine module, or 4, 8, 16, 32, and 64MB ECC DRAM on a mezzanine board. The DRAM architecture for non-ECC memory is non-interleaved for 4 or 8MB and interleaved for 16MB. Parity protection is 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 MC2chip Programming Model in the *MVME172 VME Embedded Controller Programmer's Reference Guide*.

The DRAM map decoder may be programmed to accommodate different base address(es) and sizes of mezzanine boards. The onboard DRAM is disabled by a local bus reset and must be programmed before the DRAM may be accessed. Refer to the MC2chip and MCECC descriptions in the *MVME172 VME Embedded Controller Programmer's Reference Guide* for detailed programming information.

Most DRAM devices require some number of access cycles before the DRAMs are fully operational. Normally this requirement is met by the onboard refresh circuitry and normal DRAM initialization. However, software should insure a minimum of 10 initialization cycles are performed to each bank of RAM.

SRAM Options

The MVME172 provides 128KB of 32-bit-wide onboard static RAM in a single non-interleaved architecture with onboard battery backup. The SRAM arrays are not parity protected.

The battery backup function for the onboard SRAM and the mezzanine SRAM is provided by a Dallas DS1210S device that supports primary and secondary power sources. In the event of a main board power failure, the DS1210S checks power sources and switches to the source with the higher voltage.

If the voltage of the backup source is less than two volts, the DS1210S blocks the second memory cycle; this allows software to provide an early warning to avoid data loss. Because the second access may be blocked during a power failure, software should do at least two accesses before relying on the data.

The MVME172 provides jumpers (on J14) that allow either power source of the DS1210S to be connected to the VMEbus +5V 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 +5V STDBY pin and the secondary source may be the onboard battery. If the system source should fail or the board is removed from the chassis, the onboard battery takes over.

Caution

For proper operation of the SRAM, some jumper combination must be installed on the respective Backup Power Source Select Header. Refer to the jumper information in Chapter 2. If one of the jumpers is used to select the battery, the battery must be installed on the MVME172. The SRAM may malfunction if inputs to the DS1210S are left unconnected.

The SRAM is controlled by the MC2chip, and the access time is programmable. Refer to the MC2chip description in the *MVME172 VME Embedded Controller Programmer's Reference Guide* for more detail.

About the Batteries

The power source for the onboard SRAM is a RAYOVAC FB1225 battery with two BR1225-type lithium cells. The battery is socketed for easy removal and replacement. Small capacitors are provided so that the batteries can be quickly replaced without data loss.

The lifetime of the batteries is very dependent on the ambient temperature of the board and the power-on duty cycle. The lithium batteries supplied on the MVME172 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 significantly longer 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 MVME172 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 J14 before installing the board. Refer to *SRAM Backup Power Source Select Headers (J14)* on page 2-8 for available jumper configurations

The power leads from the battery are exposed on the solder side of the board. 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 Memory

The MVME172 may be ordered with 2MB of Flash memory and two EPROM sockets ready for the installation of the EPROMs, which may be ordered separately. Flash memory is a single Intel 28F016SA device organized in a 2Mbit x 8 configuration. The EPROM locations are standard JEDEC 32-pin DIP sockets that accommodate four jumper-selectable densities (128Kbit x 8; 256 Kbit x 8; 512 Kbit x 8; 1 Mbit x8). A jumper setting (GPIO3, pins 7-8 on J11), allows reset code to be fetched either from Flash memory (GPIO3 installed) or from EPROMs (GPIO3 removed .

Battery Backed Up RAM and Clock

An M48T58 RAM and clock chip is used on the MVME172. This chip provides a time-of-day clock, oscillator, crystal, power fail detection, memory write protection, 8KB of RAM, and a battery in one 28-pin package. The clock provides seconds, minutes, hours, day, date, month, and year in BCD 24-hour format. Corrections for 28-, 29- (leap year), and 30-day months are automatically made. No interrupts are generated by the clock. Although the M48T58 is an 8 bit device, the interface provided by the MC2chip supports 8-, 16-, and 32-bit accesses to the M48T58. Refer to the MC2chip in the *MVME172 VME Embedded Controller Programmer's Reference Guide* and to the M48T58 data sheet for detailed programming and battery life information.

VMEbus Interface and VMEchip2

The VMEchip2 provides the local-bus-to-VMEbus interface, the VMEbus-to-local-bus interface, and the DMA controller functions of the local VMEbus. The VMEchip2 also provides the VMEbus system controller functions. Refer to the VMEchip2 in the *MVME172 VME 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 MC2chip ASIC at location $FFF42043. The GPI inputs are integrated into the MC2chip ASIC at location $FFF4202C bits 23-16.

I/O Interfaces

The MVME172 provides onboard I/O for many system applications. The I/O functions include serial ports and optional interfaces for IP modules, LAN Ethernet transceivers, and SCSI mass storage devices.

Serial Communications Interface

The MVME172 uses two Zilog Z85230 serial port controllers to implement the four serial communications interfaces. Each interface supports CTS, DCD, RTS, and DTR control signals, as well as the TXD and RXD transmit/receive data signals. Because the serial clocks are omitted in the MVME172 implementation, serial communications are strictly asynchronous. The MVME172 hardware supports 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 MC2chip. Refer to the Z85230 data sheet listed in this chapter, and to the MC2chip Programming Model in the *MVME172 VME Embedded Controller Programmer's Reference Guide*, for information.

The Z85230s are interfaced as DTE (data terminal equipment) with EIA-232-D signal levels. The four serial ports are routed to four RJ-45 connectors on the MVME172 front panel.

IP Interfaces

Up to two IP modules may be installed on the MVME172 as an option. The interface between the IPs and MVME172 is the IndustryPack Interface Controller (IP2) ASIC. Access to the IPs is provided by two 3M connectors located behind the MVME172 front panel. Refer to the chapter on the IP2 in the *MVME172 VME Embedded Controller Programmer's Reference Guide* for detailed features of the IP interface.

Ethernet Interface

The MVME172 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 MVME172 that has the Ethernet interface is assigned an Ethernet Station Address. The address is $08003E2*XXXXX* where *XXXXX* is the unique 5-nibble number assigned to the board (i.e., every MVME172 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, 08003E2*XXXXX* is stored in the BBRAM. The upper four bytes (08003E2*X*) are read at $FFFC1F2C; the lower two bytes (*XXXX*) are read at $FFFC1F30. The MVME162 debugger has the capability to retrieve or set the Ethernet address.

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

The Ethernet transceiver interface is located on the MVME172 main board, and the industry standard connector is located on its front panel

Support functions for the 82596CA are provided by the MC2chip. Refer to the 82596CA user's guide and to the MC2chip in the *MVME172 VME Embedded Controller Programmer's Reference Guide* for detailed programming information.

SCSI Interface

The MVME172 supports 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 MC2chip. Refer to the NCR 53C710 user's guide and to the MC2chip in the *MVME172 VME Embedded Controller Programmer's Reference Guide* for detailed programming information.

SCSI Termination

It is important that the SCSI bus is properly terminated at both ends.

The MVME172 main board provides terminators for the SCSI bus. The SCSI terminators are enabled/disabled by a jumper on header J12. If the SCSI bus ends at the MVME172, a jumper must be installed between J12 pins 1 and 2.

The FUSES LED (part of DS2) on the MVME172 front panel monitors the SCSI bus TERMPWR signal in addition to LAN power and IndustryPack power; the FUSES LED lights when all fuses are operational. The fuses are solid-state devices that reset when the short is removed.

Because any device on the SCSI bus can provide TERMPWR, the FUSES LED does not directly indicate the condition of the fuse.

Local Resources

The MVME172 includes many resources for the local processor. These include tick timers, software-programmable hardware interrupts, watchdog timer, and local bus timeout.

Programmable Tick Timers

Four 32-bit programmable tick timers with 1µs resolution are provided in the MC2chip and two 32-bit programmable tick timers are provided in the optional VMEchip2. The tick timers may be programmed to generate periodic interrupts to the processor. Refer to the VMEchip2 and MC2chip in the *MVME172 VME Embedded Controller Programmer's Reference Guide* for detailed programming information.

Watchdog Timer

A watchdog timer is provided in both the MC2chip and the optional VMEchip2. The timers operate independently but in parallel. When the watchdog timers are enabled, they must be reset

by software within the programmed time or they will time out. The watchdog timers may be programmed to generate a SYSRESET signal, local reset signal, or board fail signal if they time out. Refer to the VMEchip2 and the MC2chip in the *MVME172 VME Embedded Controller Programmer's Reference Guide* for detailed programming information.

The watchdog timer logic is duplicated in the VMEchip2 and MC2chip ASICs. Because the watchdog timer function in the VMEchip2 is a superset of that function in the MC2chip (system reset function), the timer in the VMEchip2 is used in all cases except for the version of the MVME172 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. Refer to the VMEchip2 in the *MVME172 VME Embedded Controller Programmer's Reference Guide* for detailed programming information.

Local Bus Timeout

The MVME172 provides timeout functions in the VMEchip2 and the MC2chip for the local bus. When the timer is enabled and a local bus access times out, a Transfer Error Acknowledge (TEA) signal is sent to the local bus master. The timeout value is selectable by software for 8 µsec, 64 µsec, 256 µsec, or infinite. The local bus timer does not operate during VMEbus bound cycles. VMEbus bound cycles are timed by the VMEbus access timer and the VMEbus global timer. Refer to the VMEchip2 and the MC2chip in the *MVME172 VME Embedded Controller Programmer's Reference Guide* for detailed programming information.

The MC2chip also provides local bus timeout logic for MVME172s without the optional VMEbus interface (i.e., without the VMEchip2).

The access timer logic is duplicated in the VMEchip2 and MC2chip ASICs. Because the local bus timer in the VMEchip2 can detect an offboard access and the MC2chip local bus timer cannot, the timer in the VMEchip2 is used in all cases except for the version of the MVME172 which does not include the VMEbus interface ("No VMEbus Interface option").

Local Bus Arbiter

The local bus arbiter implements a fixed priority (see Table 1-2).

Table 1-2. Local Bus Arbitration Priority

Device	Priority	Note
LAN	0	Highest
Industry Pack DMA	1	
SCSI	2	...
VMEbus	3	Next Lowest
MC68060/MC68LC060	4	Lowest

Connectors

The MVME172 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 are not used.

The MVME172 has a 20-pin connector J2 mounted behind the front panel. When the MVME172 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 MVME172 are connected to four 8-pin RJ-45 female connectors (J17) on the front panel. The two IPs connect to the MVME172 by two pairs of 50-pin connectors. Two 50-pin connectors behind the front panel are for external connections to IP

signals. The memory mezzanine board is plugged into two 100-pin connectors. The Ethernet LAN connector (J9) is a 15-pin socket connector mounted on the front panel. The SCSI connector (J10), is a 68-pin socket connector mounted on the front panel.

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 external masters (VMEbus memory map).

The memory and I/O maps that are described in the next three tables are correct for all local bus masters. There is some address translation capability in the VMEchip2. This allows multiple MVME172s 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 MVME172, 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 MC68*XX*060 MMU. The onboard I/O space must be marked cache inhibit and serialized in its page table. Table 1-4 on page 1-30 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	Notes
Programmable	DRAM on parity mezzanine	D32	4MB-16MB	N	2
Programmable	DRAM on ECC mezzanine	D32	4MB-64MB	N	2
Programmable	Onboard SRAM	D32	128KB	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
$FF800000-$FF9FFFFF	Flash/EPROM	D32	2MB	N	1, 5
$FFA00000-$FFBFFFFF	EPROM/Flash	D32	2MB	N	5
$FFC00000-$FFDFFFFF	Not decoded	D32	2MB	N	
$FFE00000-$FFE1FFFF	Onboard SRAM default	D32	128KB	N	
$FFE80000-$FFEFFFFF	Not decoded	--	512KB	N	6
$FFF00000-$FFFEFFFF	Local I/O devices (see next table)	D32-D8	878KB	Y	3

Table 1-3. Local Bus Memory Map

Address Range	Devices Accessed	Port Width	Size	Software Cache Inhibit	Notes
$FFFF0000-$FFFFFFFF	VMEbus A16	D32/D16	64KB	?	2, 4

Notes

1. Devices mapped at $FFF80000-$FFF9FFFF also appear at $00000000-$001FFFFF when the ROM0 bit in the MC2chip EPROM control register is high (ROM0=1). ROM0 is set to 1 after each reset. The ROM0 bit must be cleared before other resources (DRAM or SRAM) can be mapped in this range ($00000000 - $001FFFFF).

   The EPROM/Flash memory map is also controlled by the EPROM size and by control bit V19 in the MC2chip ASIC. Refer to the EPROM/Flash configuration tables in the *MVME172 VME Embedded Controller Programmer's Reference Guide* for further details.

2. This area is user-programmable. The DRAM and SRAM decoder is programmed in the MC2chip, the local-to-VMEbus decoders are programmed in the VMEchip2, and the IP memory space is programmed in the IP2.
3. Size is approximate.
4. Cache inhibit depends on the devices in the area mapped.
5. The EPROM and Flash are dynamically sized by the MC2chip ASIC from an 8-bit private bus to the 32-bit MPU local bus.
6. These areas are not decoded unless one of the programmable decoders is initialized to decode this space. If they are not decoded and the local timer is enabled, an access to this address range will generate a local bus timeout.

Table 1-4 describes the "Local I/O Devices" portion of the local bus main memory map.

Note The IP2 chip on the MVME172 supports up to four IP interfaces, designated IP_a through IP_d. The MVME172 itself accommodates two IPs: IP_a and IP_b. In the following map, the segments applicable to IP_c and IP_d are not used in the MVME172.

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

Address Range	Devices Accessed	Port Width	Size	Notes
$FFF00000 - $FFF3FFFF	Reserved	- -	256KB	4
$FFF40000 - $FFF400FF	VMEchip2 (LCSR)	D32	256B	1, 3
$FFF40100 - $FFF401FF	VMEchip2 (GCSR)	D32-D8	256B	1, 3
$FFF40200 - $FFF40FFF	Reserved	- -	3.5KB	4, 5
$FFF41000 - $FFF41FFF	Reserved	- -	4KB	4
$FFF42000 - $FFF42FFF	MC2chip	D32-D8	4KB	1
$FFF43000 - $FFF430FF	MCECC #1	D8	256B	1, 8
$FFF43100 - $FFF431FF	MCECC #2	D8	256B	1, 8
$FFF43200 - $FFF43FFF	MCECCs (repeated)	- -	3.5KB	1, 5, 8
$FFF44000 - $FFF44FFF	Reserved	- -	8KB	4
$FFF45000 - $FFF45800	SCC #1 (Z85230)	D8	2KB	1, 2
$FFF45801 - $FFF45FFF	SCC #2 (Z85230)	D8	2KB	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	IP2 IP_a I/O	D16	128B	1
$FFF58080 - $FFF580FF	IP2 IP_a ID	D16	128B	1

Table 1-4. Local I/O Devices Memory Map (Continued)

Address Range	Devices Accessed	Port Width	Size	Notes
$FFF58100 - $FFF5817F	IP2 IP_b I/O	D16	128B	1
$FFF58180 - $FFF581FF	IP2 IP_b ID Read	D16	128B	1
$FFF58200 - $FFF5827F	IP2 IP_c I/O	D16	128B	7
$FFF58280 - $FFF582FF	IP2 IP_c ID	D16	128B	7
$FFF58300 - $FFF5837F	IP2 IP_d I/O	D16	128B	7
$FFF58380 - $FFF583FF	IP2 IP_d ID Read	D16	128B	7
$FFF58400 - $FFF584FF	IP2 IP_ab I/O	D32-D16	256B	1
$FFF58500 - $FFF585FF	IP2 IP_cd I/O	D32-D16	256B	7
$FFF58600 - $FFF586FF	IP2 IP_ab I/O Repeated	D32-D16	256B	1
$FFF58700 - $FFF587FF	IP2 IP_cd I/O Repeated	D32-D16	256B	7
$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	IP2 Registers	D32-D8	2KB	1
$FFFBC800 - $FFFBC81F	Reserved	- -	2KB	1

Table 1-4. Local I/O Devices Memory Map (Continued)

Address Range	Devices Accessed	Port Width	Size	Notes
$FFFBD000 - $FFFBFFFF	Reserved	- -	12KB	4
$FFFC0000 - $FFFCFFFF	M48T58 (BBRAM, TOD Clock)	D32-D8	64KB	1, 9
$FFFD0000 - $FFFEFFFF	Reserved	- -	128KB	4

Notes

1. For a complete description of the register bits, refer to the data sheet for the specific chip. For a more detailed memory map, refer to the following detailed peripheral device memory maps.
2. The SCC is an 8-bit device located on an MC2chip 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. Not used.
8. To use this area, the ECC mezzanine board must be installed. If it is not installed, no acknowledge signal is returned; if the local bus timer is enabled, the access times out and is terminated by a TEA signal.
9. Repeats on 8KB boundaries.

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 that the MVME172 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 guidelines, and installation instructions for the MVME172 VME Embedded Controller.

Unpacking Instructions

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

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

Caution

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

Hardware Preparation

To select the desired configuration and ensure proper operation of the MVME172, certain option modifications may be necessary before installation. The MVME172 provides software control for most of these options. Some options cannot be modified in software, and consequently are set by installing or removing jumpers on headers. Most other modifications are performed by setting bits in control registers after the MVME172 has been installed in a system. (The MVME172 registers are described in

Chapter 4, and/or in the *MVME172 VME Embedded Controller Programmer's Reference Guide* as listed in *Related Documentation* in Chapter 1.)

Figure 2-1 illustrates the placement of the switches, jumper headers, connectors, and LED indicators on the MVME172. The MVME172 has been factory tested and is shipped with the factory jumper settings described in the following sections. The MVME172 operates with its required and factory-installed Debug Monitor, MVME172Bug (172Bug), with these factory jumper settings. Settings can be made for:

- General-Purpose Readable Jumpers Header (J21)
- General-Purpose Readable Jumpers Header (J21)
- EPROM/Flash Configuration Header (J20)
- SRAM Backup Power Source Select Headers (J14)
- SCSI Terminator Enable Header (J12)
- IP DMA Snoop Jumper (J19)
- IP Bus Strobe Select Header (J18)
- IP Bus Clock Header (J11)

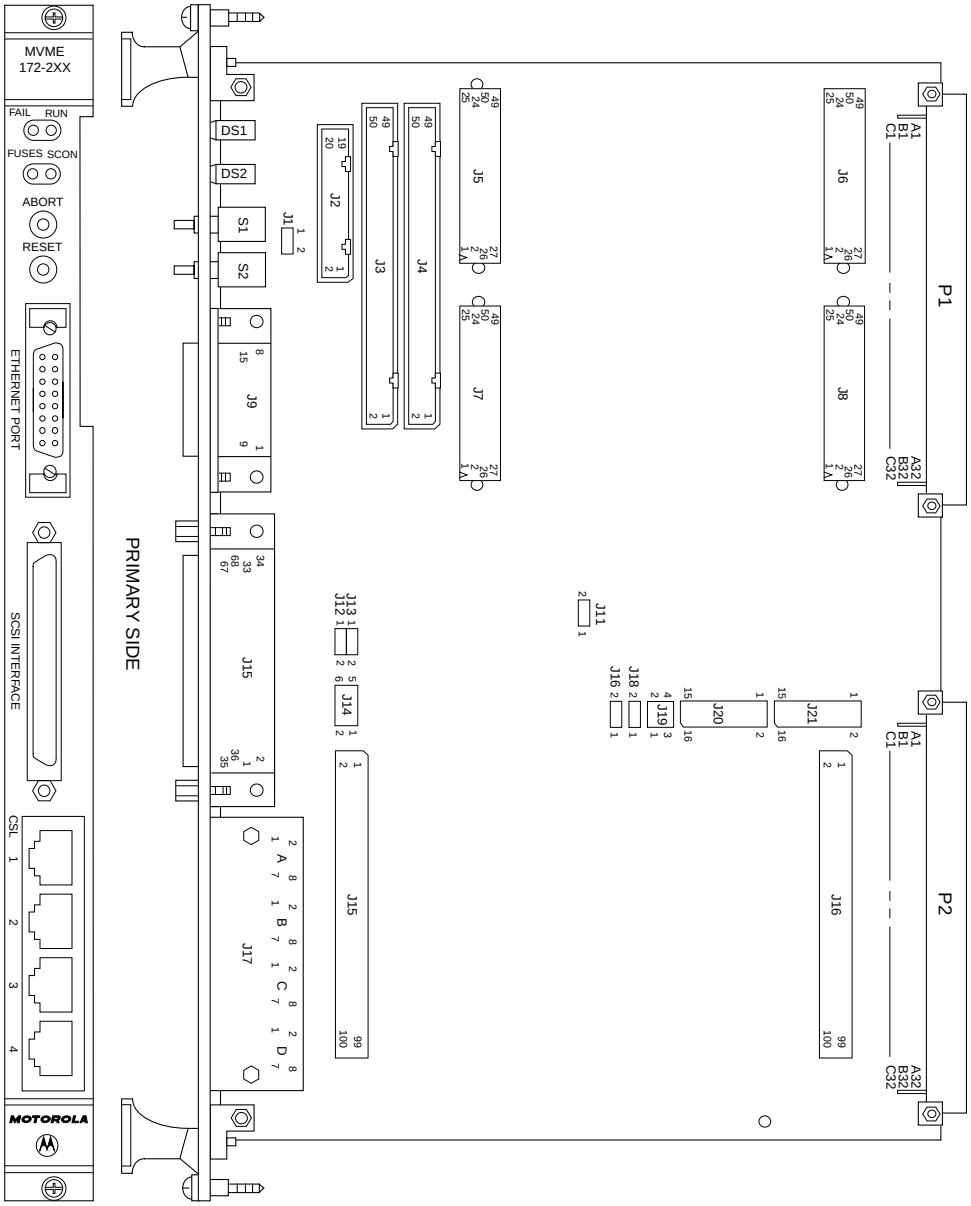
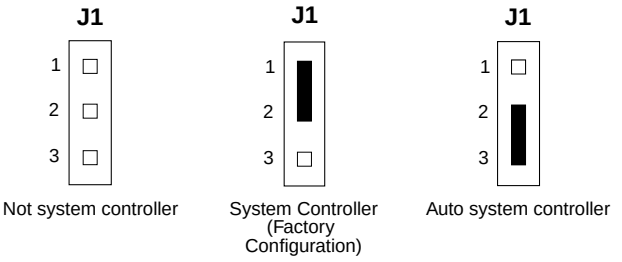


Figure 2-1. MVME172 Switch, Header, Connector, Fuse, and LED Locations

System Controller Select Header (J1)

The MVME172 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 MVME172 is not to be the system controller. Note that when the MVME172 is functioning as system controller, the SCON LED is turned on.

Note For MVME172s without the optional VMEbus interface (i.e., with no VMEchip2), the jumper may be installed or removed without affecting normal operation.



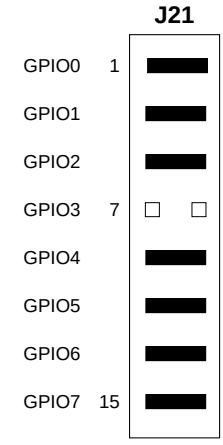
General-Purpose Readable Jumpers Header (J21)

Header J21 provides eight readable jumpers. These jumpers are read as a register (at $FFF4202D) in the MC2chip LCSR. The bit values are read as a zero when the jumper is installed, and as a one when the jumper is removed.

If the MVME172BUG firmware is installed, four jumpers are user-definable (i.e., pins 9-10, 11-12, 13-14, 15-16). If the MVME172BUG firmware is not installed, seven jumpers are user-definable (i.e., pins 1-2, 3-4, 5-6, 9-10, 11-12, 13-14, 15-16).

Note Pins 7-8 (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 address ranges for the various EPROM/Flash configurations appear in the next section of this chapter.

The MVME172 is shipped from the factory with J21 set to all zeros (jumpers on all pins) except for GPIO3.



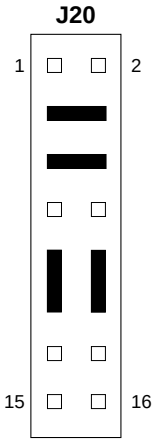
	172BUG INSTALLED	USER CODE INSTALLED
GPIO0	REFER TO 172 BUG MANUAL	USER-DEFINABLE
GPIO1	REFER TO 172BUG MANUAL	USER-DEFINABLE
GPIO2	REFER TO 172BUG MANUAL	USER-DEFINABLE
GPIO3	IN=FLASH; OUT=EPROM	IN=FLASH; OUT=EPROM
GPIO4	USER-DEFINABLE	USER-DEFINABLE
GPIO5	USER-DEFINABLE	USER-DEFINABLE
GPIO6	USER-DEFINABLE	USER-DEFINABLE
GPIO7	USER-DEFINABLE	USER-DEFINABLE

EPROMs Selected (factory configuration)

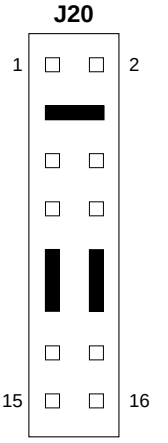
EPROM/Flash Configuration Header (J20)

The MVME172 can be ordered with 2MB of Flash memory and two EPROM sockets ready for the installation of the EPROMs, which may be ordered separately. The EPROM locations are standard JEDEC 32-pin DIP sockets that accommodate four jumper-selectable densities (128 Kbit x 8; 256 Kbit x 8; 512 Kbit x 8; 1 Mbit x 8) and permit disabling of the Flash memory.

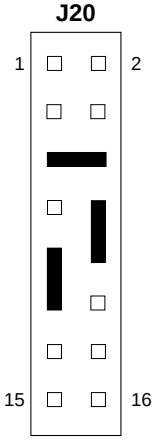
Header J20 provides eight jumpers to configure the EPROM sockets.



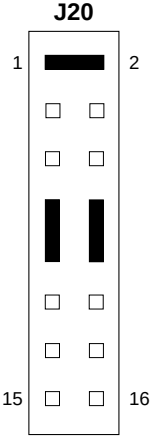
CONFIGURATION 1: 128K x 8 EPROMs



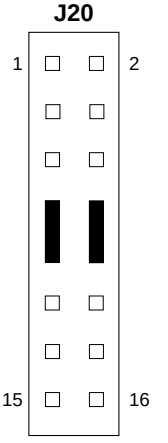
CONFIGURATION 2: 256K x 8 EPROMs



CONFIGURATION 3: 512K x 8 EPROMs



CONFIGURATION 5: 1M x 8 EPROMs
ONBOARD FLASH DISABLED



CONFIGURATION 4: 1M x 8 EPROMs

The next five tables show the address range for each EPROM socket in all five configurations. GPIO3 (J21 pins 7-8) is a control bit in the MC2chip ASIC that allows reset code to be fetched from Flash memory or from EPROMs.

Table 2-1. EPROM/Flash Mapping — 256K x 8 EPROMs

GPIO3		Address Range	Device Accessed
Removed	1	$FF800000 - $FF83FFFF	EPROM A (XU1)
		$FF840000 - $FF87FFFF	EPROM B (XU2)
		$FFA00000 - $FFBFFFFF	Onboard Flash
Installed	0	$FF800000 - $FF9FFFFF	Onboard Flash
		$FFA00000 - $FFA3FFFF	EPROM A (XU1)
		$FFA40000 - $FFA7FFFF	EPROM B (XU2)

Table 2-2. EPROM/Flash Mapping — 512K x 8 EPROMs

GPIO3		Address Range	Device Accessed
Removed	1	$FF800000 - $FF87FFFF	EPROM A (XU1)
		$FF880000 - $FF8FFFFF	EPROM B (XU2)
		$FFA00000 - $FFBFFFFF	Onboard Flash
Installed	0	$FF800000 - $FF9FFFFF	Onboard Flash
		$FFA00000 - $FFA7FFFF	EPROM A (XU1)
		$FFA80000 - $FFAFFFFF	EPROM B (XU2)

Table 2-3. EPROM/Flash Mapping — 1M x 8 EPROMs

GPIO3		Address Range	Device Accessed
Removed	1	$FF800000 - $FF8FFFFF	EPROM A (XU1)
		$FF900000 - $FF9FFFFF	EPROM B (XU2)
		$FFA00000 - $FFBFFFFF	Onboard Flash
Installed	0	$FF800000 - $FF9FFFFF	Onboard Flash
		$FFA00000 - $FFAFFFFF	EPROM A (XU1)
		$FFB00000 - $FFBFFFFF	EPROM B (XU2)

Table 2-4. EPROM/Flash Mapping — 1M x 8 EPROMs, Onboard Flash Disabled

GPIO3		Address Range	Device Accessed
Removed	1	$FF800000 - $FF8FFFFF	EPROM A (XU1)
		$FF900000 - $FF9FFFFF	EPROM B (XU2)
		Not used	Onboard Flash
Installed	0	Not used	Onboard Flash
		$FF800000 - $FF8FFFFF	EPROM A (XU1)
		$FF900000 - $FF9FFFFF	EPROM B (XU2)

SRAM Backup Power Source Select Headers (J14)

Header J14 determines the source for onboard static RAM backup power on the MVME172.

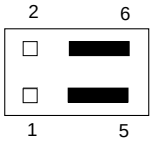
The following backup power configurations are available for onboard SRAM through header J14. In the factory configuration, the VMEbus +5V standby voltage serves as primary and secondary power source (the onboard battery is disconnected).

Note For MVME172s without the optional VMEbus interface (i.e., without the VMEchip2 ASIC), you must select the onboard battery as the backup power source.

Caution

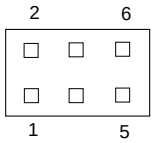
Removing all jumpers may temporarily disable the SRAM. Do not remove all jumpers from J14, except for storage.

J14



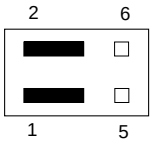
Primary Source Onboard Battery
Secondary Source Onboard Battery

J14



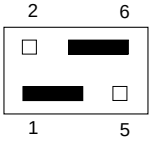
Backup Power Disabled
(For storage only)

J14



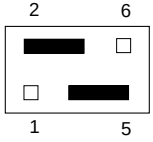
Primary Source VMEbus +5V STBY
Secondary Source VMEbus +5V STBY
(Factory configuration)

J14



Primary Source VMEbus +5V STBY
Secondary Source Onboard Battery

J14

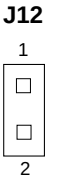


Primary Source Onboard Battery
Secondary Source VMEbus +5V STBY

SCSI Terminator Enable Header (J12)

The MVME172 provides terminators for the SCSI bus. The SCSI terminators are enabled/disabled by a jumper on header J12. The SCSI terminators may be configured as follows.

Onboard SCSI Bus Terminator Enabled
(Factory Configuration)



Onboard SCSI Bus Terminator Disabled

Note If the MVME172 is to be used at one end of the SCSI bus, the SCSI bus terminators must be enabled.

IP DMA Snoop Jumper (J19)

These jumpers control the state of the snoop signal during IP DMA. J19 pins 3 to 4 shorted inhibit snooping and 3 to 4 left unconnected enables snooping. If pins 3 to 4 are shorted, the snoop signal to the MC68060 will be driven low during IP DMA.

IP Bus Strobe Select Header (J18)

J18 pins 1 to 2 shorted connect the strobe signal from the IP2 ASCI to the IP bus. Refer to *IP Installation on the MVME172* on page 2-12.

IP Bus Clock Header (J11)

J11 selects the IP clock source. Pins 1 to 2 selects 8MHz and pins 2 to 3 selects the local bus clock which is 30MHz for the MC68060 and 32MHz for the MC68LC060. Refer to *IP Installation on the MVME172* on page 2-12.

Memory Mezzanine Options

Two 100-pin connectors (J15 and J16) are provided on the MVME172 to accommodate optional memory mezzanine boards. Two memory mezzanine options are available for the MVME172:

- 4, 8, 16MB parity DRAM
- 4, 8, 16, 32, 64MB ECC DRAM

The mezzanine boards may either be used individually or be combined in a stack (not more than two deep). The following connector options govern stacking arrangements:

- The 4, 8, and 16MB parity DRAM board has connectors on the bottom only; it must be either the only mezzanine or the top mezzanine.
- All ECC DRAM boards are available with two connector options:
  - Connectors on the bottom only; must be either the only mezzanine or the top mezzanine

When the mezzanines are stacked, the following combinations are possible:

Table 2-5. Memory Mezzanine Stacking Options

	Parity DRAM Mezzanine	**ECC DRAM Mezzanine**
Parity DRAM Mezzanine	No	Yes
ECC DRAM Mezzanine	Yes	Yes

Installation Instructions

This section covers:

- Installation of IndustryPacks (IPs) on the MVME172
- Installation of the MVME172 in a VME chassis
- System considerations relevant to the installation. Ensure that EPROM devices are installed as needed. Ensure that all header jumpers are configured as desired.

IP Installation on the MVME172

Up to two IPs may be installed on the MVME172. Install the IPs on the MVME172 as follows:

1. Each IP has two 50-pin connectors that plug into two corresponding 50-pin connectors on the MVME172: J5/J6, J7/J8. See Figure 2-1 for the MVME172 connector locations.
   – Orient the IP(s) so that the tapered connector shells mate properly. Plug IP_a into connectors J5 and J6; plug IP_b into J7 and J8. If a double-sized IP is used, plug IP_ab into J5, J6, J7, and J8.
2. Two additional 50-pin connectors (J3 and J4) are provided behind the MVME172 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., J4 has the same IP_a signals as J5; J3 has the same IP_b signals as J7).
   – Connect user-supplied 50-pin cables to J3 and J4 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 slot in the MVME172 front panel and attach them to the appropriate external equipment, depending on the nature of the particular IP(s).

MVME172 Installation

With EPROMs and IPs installed and headers properly configured, proceed as follows to install the MVME172 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 MVME172.
   – If you intend to use the MVME172 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 MVME172 as system controller, it can occupy any unused double-height card slot.
4. Slide the MVME172 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 MVME172 in the chassis with the screws provided, making good contact with the transverse mounting rails to minimize RF emissions.
6. On the chassis backplane, remove the INTERRUPT ACKNOWLEDGE (IACK) and BUS GRANT (BG) jumpers from the header for the card slot occupied by the MVME172.
7. Connect the appropriate cable(s) to the MVME172 panel connectors for the EIA-232-D serial ports, SCSI port, and LAN Ethernet port.
   – Note that some cables are not provided with the MVME172 and must be made or purchased by the user. (Motorola recommends shielded cable for all peripheral connections to minimize radiation.)
8. Connect the peripheral(s) to the cable(s).
9. Install any other required VMEmodules in the system.
10. Replace the chassis cover.
11. Connect the power cable to the AC power source and turn the equipment power ON.

System Considerations

The MVME172 draws power from both the P1 and the P2 connectors on the VMEbus backplane. P2 is also used for the upper 16 bits of data in 32-bit transfers, and for the upper 8 address lines in extended addressing mode. The MVME172 may not operate properly without its main board connected to VMEbus backplane connectors P1 and P2.

Whether the MVME172 operates as a VMEbus master or as a 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 3. D8 and/or D16 devices in

the system must be handled by the MC68060/MC68LC060 software. Refer to the memory maps in the *MVME172 VME Embedded Controller Programmer's Reference Guide*.

The MVME172 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 MVME172Bug firmware. This may be changed via software to any other base address. Refer to the *MVME172 VME Embedded Controller Programmer's Reference Guide* for more information.

If the MVME172 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 MVME172 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 MVME172 is not the system controller and there is no global bus timeout elsewhere in the system.

Multiple MVME172s may be installed in a single VME chassis. In general, hardware multiprocessor features are supported.

Note If you are installing multiple MVME172s in an MVME945 chassis, do not install an MVME172 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 MVME172 processor to broadcast a signal to any other MVME172 processors. All eight registers are accessible from any local processor as well as from the VMEbus.

The following circuits are protected by solid-state fuses that reset during overload conditions: LAN AUI, SCSI terminator, remote reset connector, IndustryPack 5V, and ±12V.

The FUSES LED illuminates to indicate that all fuses are functioning correctly. If a fuse opens, power must be removed for several minutes to allow the fuse to reset.

The MVME172 uses two Zilog Z85230 serial port controllers to implement the four serial communications interfaces. Each interface supports CTS, DCD, RTS, and DTR control signals as well as the TXD and RXD transmit/receive data signals. Because the serial clocks are omitted in the MVME172 implementation, serial communications are strictly asynchronous. The Z85230 is interfaced as DTE (data terminal equipment) with EIA-232-D signal levels. The serial ports are routed to four RJ-45 connectors on the front panel .

For additional information on the EIA-232-D interface and its implementation in the MVME172, refer to the *MVME172 VME Embedded Controller User's Manual*.

Connection diagrams for the four serial ports on the MVME172 are provided in the following figures. These ports are connected to external devices through cables connected to the front panel.

Figure 2-2 diagrams the pin assignments required in a cable to adapt a DB-25 DTE device to the RJ-45 connectors.

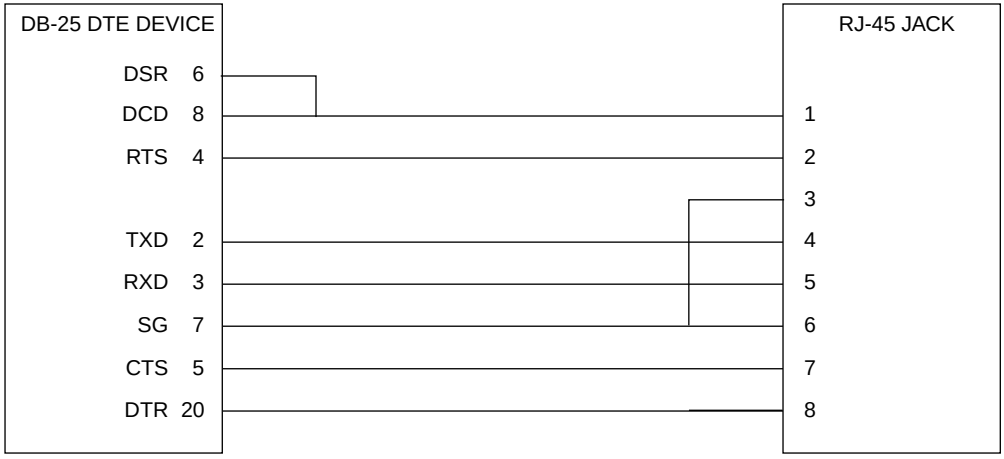


Figure 2-2. DB-25 DTE-to-RJ-45 Adapter

Figure 2-3 diagrams the pin assignments required in a cable to adapt a DB-25 DCE device to a RJ-45 connector.

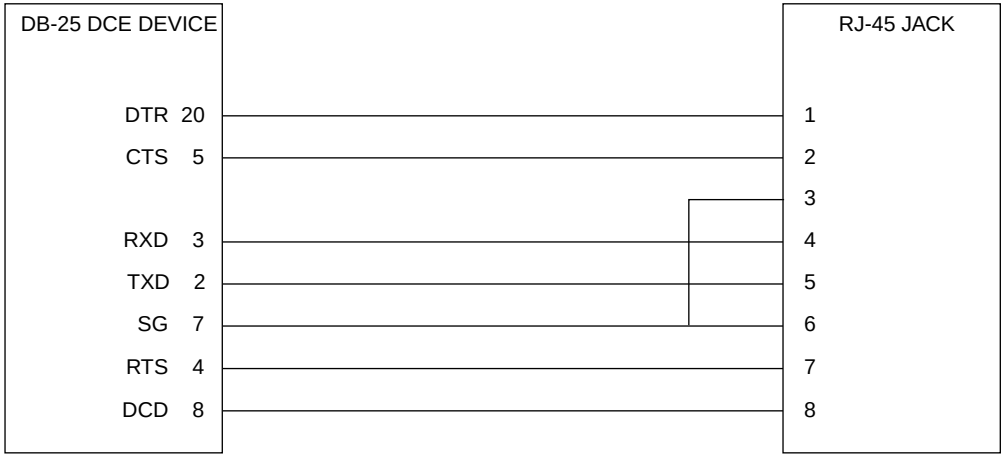


Figure 2-3. DB-25 DCE-to-RJ-45 Adapter

Figure 2-4 diagrams the pin assignments required in a typical 8-conductor serial cable having RJ-45 connectors at both ends. Note that all wires are crossed.

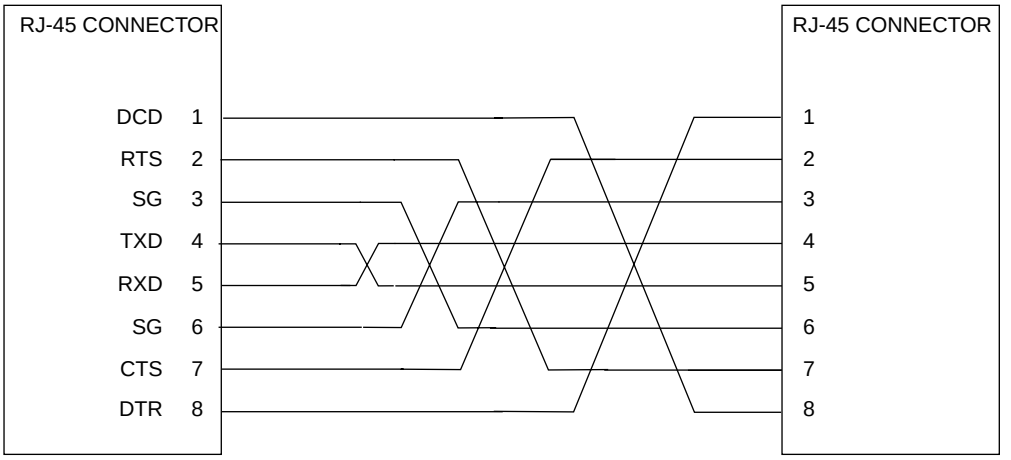


Figure 2-4. Typical RJ-45 Serial Cable

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 CPUs. 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 MVME172 MC68060- or MC68LC060-based Embedded Controller, and is known as the MVME172BUG, or 172Bug. It includes diagnostics for testing and configuring IndustryPack modules.

Description of 172Bug

172Bug is a powerful evaluation and debugging tool for systems built around the MVME172 CISC-based microcomputers. Facilities are available for loading and executing user programs under complete operator control for system evaluation. 172Bug includes commands for display and modification of memory, breakpoint and tracing capabilities, a powerful assembler/disassembler useful for patching programs, and a selftest at powerup feature which verifies the integrity of the system. Various 172Bug routines that handle I/O, data conversion, and string functions are available to user programs through the TRAP #15 system calls.

172Bug consists of three parts:

- A command-driven user-interactive software debugger, described in Chapter 4 and hereafter referred to as the debugger or 172Bug.
- A command-driven diagnostic package for the MVME172 hardware, hereafter referred to as the diagnostics.

- A user interface that accepts commands from the system console terminal.

When using 172Bug, you operate out of either the debugger directory or the diagnostic directory. If you are in the debugger directory, the debugger prompt

```
172-Bug>
```

is displayed and you have all of the debugger commands at your disposal. If you are in the diagnostic directory, the diagnostic prompt

```
172-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 172Bug is command-driven, it performs its various operations in response to user commands entered at the keyboard. When you enter a command, 172Bug executes the command and the prompt reappears. However, if you enter a command that causes execution of user target code (for example, **GO**), then control may or may not return to 172Bug, 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 172Bug 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.

3

172Bug Implementation

MVME172Bug 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, 172Bug is contained in a single 27C040 DIP EPROM installed in socket XU2, providing 512KB (128K longwords) of storage. Optionally, the 172Bug can be loaded and executed in a single Flash memory chip. 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 172Bug is installed on the MVME172, for 172Bug to operate properly with the MVME172, you must follow the steps below:

Caution

Inserting or removing boards while power is applied could damage board components.

1. Turn all equipment power OFF. Refer to the *Hardware Preparation* on page 2-1 and install/remove jumpers on headers as required for your particular application.

   Jumpers on header J21 affect 172Bug operation as described below. The default condition for the MVME172 is with seven jumpers installed, between pins 1-2, 3-4, 5-6, 9-10, 11-12, 13-14, and 15-16 (no jumper between pins 7-8).

These readable jumpers are read as a register (at $FFF4202D) on the MC2chipASIC. The bit values are read as a zero when the jumper is installed, and as a one when the jumper is removed. This jumper block (header J21) contains eight bits. Refer also to the *MVME172 VME Embedded Controller Programmer's Reference Guide* for more information on the MC2chip.

The MVME172Bug reserves/defines the four lower order bits (GPI3 to GPI0). The following is the description for the bits reserved/defined by the debugger:

Bit	J21 Pins	Description
Bit #0 (GPI0)	1-2	When this bit is a one (high), it instructs the debugger to use local Static RAM for its work page (i.e., variables, stack, vector tables, etc.).
Bit #1 (GPI1)	3-4	When this bit is a one (high), it instructs the debugger to use the default setup/operation parameters in ROM versus the user setup/operation parameters in Non-Volatile RAM (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/ROM defaults.
Bit #2 (GPI2)	5-6	Reserved for future use.
Bit #3 (GPI3)	7-8	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)	9-10	Open to your application.
Bit #5 (GPI5)	11-12	Open to your application.
Bit #6 (GPI6)	13-14	Open to your application.
Bit #7 (GPI7)	15-16	Open to your application.

3

Note that when the MVME172 comes up in a cold reset, 172Bug runs in Board Mode. Using the Environment (**ENV**) or **MENU** commands can make 172Bug 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 MVME172.
3. Refer to the setup procedure for your particular chassis or system for details concerning the installation of the MVME172.
4. Connect the terminal that is to be used as the 172Bug system console to the default debug EIA-232-D port at serial port 1 on the front panel of the MVME172. 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 MVME172 ports at powerup)

   After powerup, the baud rate of the debug port can be reconfigured by using the Port Format (**PF**) command of the 172Bug debugger.

Note In order for high-baud rate serial communication between 172Bug and the terminal to work, the terminal must do some form of handshaking. If the terminal being used does not do hardware handshaking via the CTS line, then it must do XON/XOFF handshaking. If you get garbled messages and missing characters, then you should check the terminal to make sure XON/XOFF handshaking is enabled.

5. If you want to connect devices (such as a host computer system and/or a serial printer) to the other EIA-232-D port connectors, connect the appropriate cables and configure the port(s) as detailed in step 4 above. After powerup, this (these) port(s) can be reconfigured by programming the MVME172 Z85230 Serial Communications Controllers (SCCs), or by using the 172Bug **PF** command.

6. The EPROM/Flash header J20 must be set to configuration 3, with jumpers between J20 pins 5 and 6, 8 and 10, and 9 and 11. This sets it up for 512K x 8 EPROMs.

7. Power up the system. 172Bug executes some self-checks and displays the debugger prompt

   ```
   172-Bug>
   ```

   (if 172Bug is in Board Mode). However, if the **ENV** command (Appendix A) has put 172Bug in System Mode, the system performs a selftest and tries to autoboot. Refer to the **ENV** and **MENU** commands (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.

8. Before using the MVME172 after the initial installation, set the date and time using the following command line structure:

   172-Bug> **SET** [*mmddyyhhmm*] | [<+/-*CAL*>;**C**]

   For example, the following command line starts the real-time clock and sets the date and time to 10:37 a.m., July 7, 1997:

   172-Bug> **SET 0707971037**

   The board's self-tests and operating systems require that the real-time clock be running.

Autoboot

3

Autoboot is a software routine that is contained in the 172Bug 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 powerup, 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 MVME172. At powerup, the tape controller will position the streaming tape to load point where the volume ID can correctly be read and used.

If, however, the MVME172 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, 172Bug occupies an EPROM installed in socket XU2. This leaves one socket (XU1) and the Flash available for your use. Contact your Motorola sales office for assistance. This function is configured/enabled by the Environment (**ENV**) command (refer to Appendix A) and executed at powerup (optionally also at reset) or by the **RB** command assuming there is valid code in the memory devices (or optionally elsewhere on the board or VMEbus) to support it. If ROMboot code is installed, a user-written routine is given control (if the routine meets the format requirements). One use of ROMboot might be resetting SYSFAIL* on an unintelligent controller module. The **NORB** command disables the function.

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

- Power must have just been applied (but the **ENV** command can change this to also respond to any reset).
- Your routine must be located within the MVME172 Flash/PROM memory map (but the **ENV** command can change this to any other portion of the onboard memory, or even offboard VMEbus memory).
- The ASCII string "BOOT" must be located within the specified memory range.

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

3

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 172Bug 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 powerup, 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 MVME172 front panel RESET switch initiates a system reset. COLD and WARM reset modes are available. By default, 172Bug is in COLD mode. During COLD reset, a total system initialization takes place, as if the MVME172 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 172Bug 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 172Bug environment is ever lost (vector table is destroyed, stack corrupted, etc.).

Abort

Abort is invoked by pressing and releasing the ABORT switch on the MVME172 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 "power-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

3

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

3

The program portion of 172Bug is approximately 512KB of code, consisting of download, debugger, and diagnostic packages and contained entirely in Flash or PROM.

The 172Bug executes from $FF800000 whether in Flash or PROM. If you remove the jumper at J21 pins 7 and 8, the address spaces of the Flash and PROM are swapped. For the MVME172-2*XX* (MVME172), factory ship is with jumper J21 pins 7-8 removed (172Bug operates out of EPROM).

The 172Bug initial stack completely changes 8KB of SRAM memory at addresses offset $C000 from the SRAM base address, 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 172Bug requires 2KB of NVRAM for storage of board configuration, communication, and booting parameters. This storage area begins at $FFFC16F8 and ends at $FFFC1EF7.

172Bug 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 172Bug stack and static variable space and the rest is reserved as user space. Whenever the MVME172 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.

Disk I/O Support

172Bug can initiate disk input/output by communicating with intelligent disk controller modules over the VMEbus. Disk support facilities built into 172Bug 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 172Bug. Default values for these parameters are assigned at powerup and cold-start reset, but may be altered as described in the section on default parameters, later in this chapter.

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 172Bug 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.

3

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. 172Bug 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 172Bug Commands

These following 172Bug 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 172Bug 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 172Bug. It is used as a debugging tool.

Disk I/O via 172Bug System Calls

All operations that actually access the disk are done directly or indirectly by 172Bug 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, 172Bug 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 172Bug Controller and Device Parameters

172Bug 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 two 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.

Disk I/O Error Codes

172Bug returns an error code if an attempted disk operation is unsuccessful.

Network I/O Support

3

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

172Bug returns an error code if an attempted network operation is unsuccessful.

Multiprocessor Support

The MVME172 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 MVME172 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:

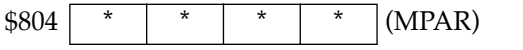
HEX 0 (HEX 00) - Wait. Initialization not yet complete.
ASCII E (HEX 45) - Code pointed to by the MPAR address is executing.
ASCII P (HEX 50) - Program Flash Memory. The MPAR is set to the address of the Flash memory program control packet.
ASCII R (HEX 52) - Ready. The firmware monitor is watching for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

You can only program Flash memory by the MPCR method. Refer to the .PFLASH system call in the *MVME172Bug 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 powerup, the debug monitor selftest routines initialize RAM, including the memory locations used for multi-processor support ($800 through $807).

The MPCR contains $00 at powerup, 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 MVME172 dual-port RAM by issuing a remote **GO** command using the VMEchip2 Global Control and Status Registers (GCSR). The remote processor places the MVME172 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 MVME172 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 MVME172. 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 172Bug package includes a set of hardware diagnostics for testing and troubleshooting the MVME172. 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

```
172-Diag>
```

appears. Refer to the *MVME172Bug 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 MVME172s, 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 board 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.

Using the 172Bug Debugger 4

This Chapter Covers

- Entering debugger command lines
- Entering and debugging programs
- Calling system utilities from user programs
- Preserving the debugger operating environment
- Floating point support
- The 172Bug debugger command set

Entering Debugger Command Lines

172Bug is command-driven and performs its various operations in response to user commands entered at the keyboard. When the debugger prompt

```
172-Bug>
```

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

Terminal Input/Output Control

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

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

^X	(cancel line)	The cursor is backspaced to the beginning of the line.
^H	(backspace)	The cursor is moved back one position.
Delete key	(delete)	Performs the same function as **^H**.
^D	(redisplay)	The entire command line as entered so far is redisplayed on the following line.
^A	(repeat)	Repeats the previous line. This happens only at the command line. The last line entered is redisplayed but not executed. The cursor is positioned at the end of the line. You may enter the line as is or you can add more characters to it. You can edit the line by backspacing and typing over old characters.

When observing output from any 172Bug 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 172Bug, but you may change them with the **PF** command. In the initialized (default) mode, operation is as follows:

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

When a command is entered, the debugger executes the command and the prompt reappears. However, if the command entered causes execution of user target code, for example **GO**, then control may or may not return to the debugger, depending on what the user program does.

For example, if a breakpoint has been specified, then control returns to the debugger when the breakpoint is encountered during execution of the user program. Alternately, the user program could return to the debugger by means of the TRAP #15 function ".RETURN".

4

Debugger Command Syntax

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

- The command identifier (i.e., **MD** or **md** for the Memory Display command). Note that either upper- or lowercase is allowed.
- A port number if the command is set up to work with more than one port.
- At least one intervening space before the first argument.
- Any required arguments, as specified by the command.
- An option field, set off by a semicolon (**;**) to specify conditions other than the default conditions of the command.

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

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.

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 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 (>>).

4

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

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

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

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

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

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

Valid expression examples:

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

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

4

Address as a Parameter

Many commands use *addr* as a parameter. The syntax accepted by 172Bug is similar to the one accepted by the MC68060 one-line assembler. All control addressing modes are allowed. An "address + offset register" mode is also provided.

Address Formats

Table 4-1 summarizes the address formats that 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+R*n*	130+R5	Absolute address+contents of the specified offset register (not an assembler-accepted syntax).
(A*n*)	(A1)	Address register indirect. (Also post-increment, pre-decrement)
(*d*,A*n*) or *d*(A*n*)	(120,A1) 120(A1)	Address register indirect with displacement (two formats accepted).

Table 4-1. Debugger Address Parameter Formats

Format	Example	Description
(*d*,A*n*,*Xn*) or *d*(A*n*,*Xn*)	(&120,A1,D2) &120(A1,D2)	Address register indirect with index and displacement (two formats accepted).
([*bd*,A*n*,*Xn*],*od*)	([C,A2,A3],&100)	Memory indirect preindexed.
([*bd*,A*n*],X*n*,*od*)	([12,A3],D2,&10)	Memory indirect postindexed.
For the memory indirect modes, fields can be omitted. For example, three of many permutations are as follows:		
([,A*n*],*od*)	([,A1],4)	
([*bd*])	([FC1E])	
([*bd*,,*Xn*])	([8,,D2])	

Notes

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

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

Offset Registers

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

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

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

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

The above program was loaded at address $0001327C.

The disassembled code is shown next:

```
172Bug>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
172Bug>
```

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

```
172Bug>OF R0
R0 =00000000 00000000? 1327C. <CR>
172Bug>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
172Bug>
```

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

Port Numbers

Some 172Bug 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. MVME172 EIA-232-D Debug (Terminal Port 0 or 00) (PORT 1 on the MVME172 J17 front panel connector). Sometimes known as the "console port", it is used for interactive user input/output by default.
2. MVME172 EIA-232-D (Terminal Port 1 or 01) (PORT 2 on the MVME172 J17 front panel connector). Sometimes known as the "host port", this is the default for downloading, uploading, concurrent mode, and transparent modes.
3. MVME172 EIA-232-D (Terminal Ports 2 or 02 and 3 or 03) (PORT 3 and PORT 4 on the MVME172 J17 front panel connector). Additional serial ports available.

Note These logical port numbers (0, 1, 2, and 3) are shown in the pinouts of the MVME as "SERIAL PORT 1", "SERIAL PORT 2", "SERIAL PORT 3", and "SERIAL PORT 4", respectively. Physically, they are all part of the front panel 8-pin RJ-45 connectors on J17.

Entering and Debugging Programs

There are various ways to enter a user program into system memory for execution:

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

Creating a Program with the Assembler/Disassembler

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

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

Refer to the *Debugging Package for Motorola 68K CISC CPUs User's Manual* for complete details of the 172Bug Assembler/Disassembler.

Downloading an S-Record Object File

Another way to enter a program is to download an object file from a host system.

The program must be in S-record format (described in the *Debugging Package for Motorola 68K CISC CPUs User's Manual*) and may have been assembled or compiled on the host system.

Alternately, the program may have been previously created using the 172Bug **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 MVME172 port 1. (Hardware configuration details are provided in *Installation and Startup* on page 3-3.) The file is downloaded from the host to MVME172 memory by the Load (**LO**) command.

Read the Program from Disk

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

Calling System Utilities from User Programs

A convenient way of doing character input/output and many other useful operations has been provided so that you do not have to write these routines into the target code. You can access various 172Bug routines via one of the MC68060 TRAP instructions, using vector #15. Refer to the *Debugging Package for Motorola 68K CISC CPUs User's Manual* for details on the various TRAP #15 utilities available and how to invoke them from within a user program.

Preserving the Debugger Operating Environment

This section explains how to avoid contaminating the operating environment of the debugger. Topics covered include:

4

- 172Bug Vector Table and workspace
- Hardware functions
- Exception vectors used by 172Bug

172Bug uses certain of the MVME172 onboard resources and may also use offboard system memory to contain temporary variables, exception vectors, etc. If you disturb resources upon which 172Bug 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.

172Bug Vector Table and Workspace

As described in *Memory Requirements* on page 3-13, 172Bug needs 64KB of read/write memory to operate.

172Bug reserves ...	For ...
1024-byte area	A user program vector table area
1024-byte area	An exception vector table for the debugger itself to use
Space for static variables and initializes these static variables to predefined default values.	
Space for the system stack then initializes the system stack pointer to the top of this area.	

With the exception of the first 1024-byte vector table area, you must be extremely careful not to use the above-mentioned memory areas for other purposes.

Refer to *Memory Requirements* on page 3-13 to determine how to dictate the location of the reserved memory areas.

4

Examples

- If, for example, your program inadvertently wrote over the static variable area containing the serial communication parameters, these parameters would be lost, resulting in a loss of communication with the system console terminal.
- If your program corrupts the system stack, then an incorrect value may be loaded into the processor Program Counter (PC), causing a system crash.

Hardware Functions

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

Exception Vectors Used by 172Bug

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 172Bug

Vector Offset	Exception	172Bug 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 MC2chip.
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.

```
172-Bug>rd
PC     =00010000    SR     =2708=TR:OFF_S._7_N..            VBR    =00000000
SSP    =0000FFFC    USP    =00010000    SFC     =1=UD       DFC    =1=UD
CACR   =00000000=D: ....._B:..._I:...                       PCR    =04310402
D0     =FFFFFFFF    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
IPLR   =00000007    IMLR   =00000000    MMIEN   =00000003   VIEN   =C0000000
VIST   =00000000    PIEN   =00002000    PIST    =00000000
00010000 203C0000 0001     MOVE.L    #$1,D0
172-Bug>t
PC     =00010006    SR     =2700=TR:OFF_S._7_.....          VBR    =00000000
SSP*   =0000FFFC    USP    =00010000    SFC     =1=UD       DFC    =1=UD
CACR   =00000000=D: ....._B:..._I:...                       PCR    =04310402
D0     =F0000001    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
IPLR   =00000007    IMLR   =00000000    MMIEN   =00000003   VIEN   =C0000000
VIST   =00000000    PIEN   =00002000    PIST    =00000000
00010006 D280 0001         ADD.L      D0,D1
172-Bug>
```

Exception Vector Tables

Notice in the preceding example that the value of the target stack pointer register (A7) has not changed even though a trace exception has taken place. Your program may either use the exception vector table provided by 172Bug or it may create a separate exception vector table of its own. The two following sections detail these two methods.

Using 172Bug Target Vector Table

The 172Bug initializes and maintains a vector table area for target programs. A target program is any program started by the bug:

- Manually with **GO** command
- Manually with trace commands (**T**, **TC**, or **TT**)
- Automatically with the **BO** command.

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

The 172Bug initializes the target vector table with the debugger vectors listed in Table 4-2 on page 4-15 and fills the other vector locations with the address of a generalized exception handler. 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 172Bug 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 172Bug vector table into the corresponding vector locations in your program vector table.

The vector for the 172Bug generalized exception handler 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 172Bug target program vector table (which your program saved), yielding the address of the 172Bug exception vector. The program then jumps to the address stored at this vector location, which is the address of the 172Bug 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:

4

```
*
***  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.
```

Floating Point Support

The floating point unit (FPU) of the MC68060 microprocessor chip is supported in 172Bug. The **MD**, **MM**, **RM**, and **RS** commands have been extended to allow display and modification of floating point data in registers and in memory. Floating point instructions can be assembled and disassembled with the **DI** option of the **MD** and **MM** commands.

RM and **RS** for floating point registers accept the floating point value in Double Precision Real Format or Scientific Notation.

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
-3.12345678901234501_E+123	Scientific Notation Format (decimal)

When entering data in single or double precision, 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).

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.

4

Entering data in this format requires the following fields:

An optional sign bit (+ or -).
One decimal digit followed by a decimal point.
Up to 17 decimal digits (at least one must be entered).
An optional Exponent field that consists of:

- An optional underscore.
- The Exponent field identifier, letter “E”.
- An optional Exponent sign (+, -).
- From 1 to 3 decimal digits.

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

The 172Bug Debugger Command Set

4

The 172Bug 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 MVME*172Bug 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 addr* [**; B** | **W** | **L**]
BF	Block of Memory Fill	**BF** *range data* [*increment*] [**; B** | **W** | **L**]
BH	Bootstrap Operating System and Halt	**BH** [*controller LUN*] [*device LUN*] [*string*]
BI	Block of Memory Initialize	**BI** *range* [**; B** | **W** | **L**]
BM	Block of Memory Move	**BM** *range addr* [**; B** | **W** | **L**]
BO	Bootstrap Operating System	**BO** [*controller LUN*] [*device LUN*] [*string*]
BR	Breakpoint Insert	**BR** [*addr* [:*count*]]
NOBR	Breakpoint Delete	**NOBR** [*addr*]
BS	Block of Memory Search	**BS** *range text* [**; B** | **W** | **L**] or **BS** *range data* [*mask*] [**; B** | **W** | **L** [**,N**] [**,V**]]
BV	Block of Memory Verify	**BV** *range data* [*increment*] [**; B** | **W** | **L**]
CM	Concurrent Mode	**CM** [[*port*] [*ID-string] [baud*] [*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**]]

Table 4-3. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
DMA	DMA Block of Memory Move	**DMA** *range addr vdir am blk* [**; B** | **W** | **L**]
DS	One Line Disassembler	**DS** *addr* [**:***count* | *addr*]
DU	Dump S-records	**DU** [*port*] *range* [*text*] [*addr*] [*offset*] [**; B** | **W** | **L**]
ECHO	Echo String	**ECHO** [*port*] {*hexadecimal number*} {'*string*'}
ENV	Set Environment to Bug/Operating System	**ENV** [**; [D**]]
GD	Go Direct (Ignore Breakpoints)	**GD** [*addr*]
GN	Go to Next Instruction	**GN**
GO	Go Execute User Program	**GO** [*addr*]
GT	Go to Temporary Breakpoint	**GT** *addr*
HE	Help	**HE** [*command*]
IOC	I/O Control for Disk	**IOC**
IOI	I/O Inquiry	**IOI** [**; [C** | **L**]]
IOP	I/O Physical (Direct Disk Access)	**IOP**
IOT	I/O "TEACH" for Configuring Disk Controller	**IOT** [**; [A** | **F** | **H** | **T**]]
IRQM	Interrupt Request Mask	**IRQM** [*mask*]
LO	Load S-records from Host	**LO** [*port*] [*addr*] [**; [X] [C] [T]**] [*=text*]
MA	Macro Define/Display	**MA** [*name* | **; L**]
NOMA	Macro Delete	**NOMA** [*name*]
MAE	Macro Edit	**MAE** *name line#* [*string*]
MAL	Enable Macro Expansion Listing	**MAL**
NOMAL	Disable Macro Expansion Listing	**NOMAL**
MAW	Save Macros	**MAW** [*controller LUN*] [*device LUN*] [*del block #*]

Table 4-3. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
MAR	Load Macros	**MAR** [*controller LUN*] [*device LUN*] [*del block #*]
MD	Memory Display	**MD** [**S**] *addr* [**:***count* | *addr*] [**; [B | W | L | S | D | DI]**]
MENU	Menu	**MENU**
MM	Memory Modify	**MM** *addr* [**; [[B | W | L | S | D] [A] [N]] | [DI]]**
MMD	Memory Map Diagnostic	**MMD** *range 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** [*port*]
NOPA	Printer Detach	**NOPA** [*port*]
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**

Table 4-3. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
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*]
RS	Register Set	**RS** *reg* [*exp* | *addr*]
SD	Switch Directories	**SD**
SET	Set Time and Date	**SET** *mmddyyhhmm* | *n***; C**
SYM	Symbol Table Attach	**SYM** [*addr*]
NOSYM	Symbol Table Detach	**NOSYM**
SYMS	Symbol Table Display/Search	**SYMS** [*symbol-name*] | [**; S**]
T	Trace	**T** [*count*]
TA	Terminal Attach	**TA** [*port*]
TC	Trace on Change of Control Flow	**TC** [*count*]
TIME	Display Time and Date	**TIME** [**;** [**C** | **L** | **O**]]
TM	Transparent Mode	**TM** [*port*] [ESCAPE]
TT	Trace to Temporary Breakpoint	**TT** *addr*
VE	Verify S-Records Against Memory	**VE** [*port*] [*addr*] [**;** [**X**][**C**]] [=*text*]
VER	Display Revision/Version	**VER** [**; E**]
WL	Write Loop	**WL** *addr 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 *MVME172 VME Embedded Controller User's Manual* for the actual location. The information block contains various elements detailing specific operation parameters of the hardware. The *MVME172 VME 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

The board structure for the MVME172 is as follows:

```
172-Bug>cnfg
Board (PWA) Serial Number = "     "
Board Identifier = "     "
Artwork (PWA) Identifier = "     "
MPU Clock Speed = "     "
Ethernet Address = 08003E200000
Local SCSI Identifier = "     "
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 = "     "
172-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 Table A-1:

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 MPU 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 negotiations.
Industry Pack Reset on Debugger Startup [Y/N]	Y	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.

Table A-1. ENV Command Parameters (Continued)

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

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Network Auto Boot Enable [Y/N]	N	Network Auto Boot function is disabled.
Network Auto Boot at power-up only [Y/N]	Y	Network Auto Boot is attempted at power up reset only.
Network Auto Boot Controller LUN	00	LUN of a disk/tape controller module currently supported by the Bug. Default is $0.
Network Auto Boot Device LUN	00	LUN of a disk/tape device currently supported by the Bug. Default is $0.
Network Auto Boot Abort Delay	5	This is the time in seconds that the Network Boot sequence will delay before starting the boot. The purpose for the delay is to allow you the option of stopping the boot by use of the Break key. The time value is from 0 through 255 seconds.
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-172 environment, each MVME172 board could be set to start its work page at a unique address to allow multiple debuggers to operate simultaneously.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
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 MVME172. Default Memory Search Ending Address is the calculated size of local memory.
Memory Search Increment Size	00010000	This multi-CPU feature is used to offset the location of the Bug work page. This must be a multiple of the debugger work page, modulo $10000 (64KB). Typically, Memory Search Increment Size is the product of CPU number and size of the Bug work page. Example: first CPU $0 (0 x $10000), second CPU $10000 (1 x $10000), etc.
Memory Search Delay Enable [Y/N]	N	There will be no delay before the Bug begins its search for a work page.

A

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Memory Search Delay Address	FFFFD20F	Default address is $FFFFD20F. This is the MVME172 GCSR GPCSR0 as accessed through VMEbus A16 space and assumes the MVME172 GRPAD (group address) and BDAD (board address within group) switches are set to "on". This byte-wide value is initialized to $FF by MVME172 hardware after a System or Power-on Reset. In a multi-172 environment, where the work pages of several Bugs will reside in the memory of the primary (first) MVME172, 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.

Note

Memory Configuration Defaults.
The default configuration for Dynamic RAM mezzanine boards will position the mezzanine with the largest memory size to start at the address selected with the "ENV" parameter "Base Address of Dynamic Memory". The Base Address parameter defaults to 0. The smaller sized mezzanine will follow immediately above the larger in the memory map. If mezzanines of the same size and type are present, the first (closest to the board) is mapped to the selected base address. If mezzanines of same size but with different type (parity and ECC) are present, the parity type will be mapped to the selected base address and the ECC type mezzanine will follow. The SRAM does not default to a location in the memory map that is contiguous with Dynamic RAM.

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 is FFE00000 for the onboard 128KB SRAM, or E1000000 for the 2MB SRAM mezzanine. If only 2MB SRAM is present, it defaults to address 00000000.
Size of 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 MVME172 series modules. You should have a working knowledge of the VMEchip2 as given in the *MVME172 VME 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 MVME172. There are two slave address decoders set. They are set up as follows:		
Slave Enable #1 [Y/N]	Y	Yes, setup and enable the Slave Address Decoder #1.

A

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Slave Starting Address #1	00000000	Base address of the local resource that is accessible by the VMEbus. Default is the base of local memory, $0.
Slave Ending Address #1	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 setup 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.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Master Enable #1 [Y/N]	Y	Yes, setup and enable the Master Address Decoder #1.
Master Starting Address #1	02000000	Base address of the VMEbus resource that is accessible from the local bus. Default is the end of calculated local memory, unless memory is less than 16MB, then this register will always be set to 01000000.
Master Ending Address #1	EFFFFFFF	Ending address of the VMEbus resource that is accessible from the local bus. Default is the end of calculated memory.
Master Control #1	0D	Defines the access characteristics for the address space defined with this master address decoder. Default is $0D.
Master Enable #2 [Y/N]	N	Do not setup and enable the Master Address Decoder #2.
Master Starting Address #2	00000000	Base address of the VMEbus resource that is accessible from the local bus. Default is $00000000.
Master Ending Address #2	00000000	Ending address of the VMEbus resource that is accessible from the local bus. Default is $00000000.
Master Control #2	00	Defines the access characteristics for the address space defined with this master address decoder. Default is $00.
Master Enable #3 [Y/N]	N	Yes, setup and enable the Master Address Decoder #3. This is the default if the board contains less than 16MB of calculated RAM. Do not setup 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.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
Master Control #4	00	Defines the access characteristics for the address space defined with this master address decoder. Default is $00.
Short I/O (VMEbus A16) Enable [Y/N]	Y	Yes, Enable the Short I/O Address Decoder.
Short I/O (VMEbus A16) Control	01	Defines the access characteristics for the address space defined with the Short I/O address decoder. Default is $01.
F-Page (VMEbus A24) Enable [Y/N]	Y	Yes, Enable the F-Page Address Decoder.
F-Page (VMEbus A24) Control	02	Defines the access characteristics for the address space defined with the F-Page address decoder. Default is $02.
ROM Access Time Code	04	This defines the ROM access time. The default is $04, which sets an access time of five clock cycles of the local bus.
Flash Access Time Code	03	This defines the Flash access time. The default is $03, which sets an access time of four clock cycles of the local bus.
MCC Vector Base VMEC2 Vector Base #1 VMEC2 Vector Base #2	05 06 07	Base interrupt vector for the component specified. Default: MC2chip = $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.

Table A-1. ENV Command Parameters (Continued)

ENV Parameter and Options	Default	Meaning of Default
VMEC2 GCSR Board Base Address	00	Specifies the base address ($FFFFD2*XX*) in Short I/O for this board. Default = $00.
VMEbus Global Time Out Code	01	This controls the VMEbus timeout when the MVME172 is 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 MVME172s.

The *MVME172 VME 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.

Note The IP2 ASIC on the MVME172 supports up to four IndustryPack (IP) interfaces, designated IP_a through IP_d. The MVME172 itself accommodates two IPs: IP_a and IP_b. In the following discussion, the segments applicable to IP_c and IP_d are not used in the MVME172.

```
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

A

```
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

Caution

Before environment parameters are saved in the NVRAM, a warning message will appear if the user has specified environment parameters which will cause an overlap condition. The important information about each configurable element in the memory map is displayed, showing where any overlap conditions exist. This will allow the user to quickly identify and correct an undesirable configuration before it is saved.

ENV warning example:

```
WARNING: Memory MAP Overlap Condition Exists

S-Address    E-Address    Enable  Overlap  M-Type   Memory-MAP-Name
$00000000    $FFFFFFFF    Yes     Yes      Master   Local Memory (Dynamic RAM)
$FFE00000    $FFE7FFFF    Yes     Yes      Master   Static RAM
$01000000    $EFFFFFFF    Yes     Yes      Master   VMEbus Master #1
$00000000    $00000000    No      No       Master   VMEbus Master #2
$00000000    $00FFFFFF    Yes     Yes      Master   VMEbus Master #3
$00000000    $00000000    No      No       Master   VMEbus Master #4
$F0000000    $FF7FFFFF    Yes     Yes      Master   VMEbus F Pages (A24/A32)
$FFFF0000    $FFFFFFFF    Yes     Yes      Master   VMEbus Short I/O (A16)
$FF800000    $FFBFFFFF    Yes     Yes      Master   Flash/PROM
$FFF00000    $FFFEFFFF    Yes     Yes      Master   Local I/O
$00000000    $00000000    No      No       Master   Industry Pack A
$00000000    $00000000    No      No       Master   Industry Pack B
$00000000    $00000000    No      No       Master   Industry Pack C
$00000000    $00000000    No      No       Master   Industry Pack D
$00000000    $00000000    No      No       Slave    VMEbus Slave #1
$00000000    $00000000    No      No       Slave    VMEbus Slave #2
```

Disk/Tape Controller Data

B

Disk/Tape Controller Modules Supported

The following VMEbus disk/tape controller modules are supported by the 172Bug. 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*	--	--	--
MVME328 - SCSI Controller	$06	$FFFF9000	$07	$FFFF9800
MVME328 - SCSI Controller	$16	$FFFF4800	$17	$FFFF5800
MVME328 - SCSI Controller	$18	$FFFF7000	$19	$FFFF7800

*If an MVME172 with a SCSI port is used, then the MVME172 has CLUN 0.

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

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		

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 MVME172.

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

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

Notes

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

Network Controller Modules Supported

The following VMEbus network controller modules are supported by the MVME172BUG. The default address for each type and position is showed to indicate where the controller must reside to be supported by the MVME172BUG. 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
MVME172	$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

C

Index

D

E

F

G

H

I

J

L

M

N

INDEX