MVME147

MPU VMEmodule

Installation and Use

VME147A/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.

No part of this material may be reproduced or copied in any tangible medium, or stored in a retrieval system, or transmitted in any form, or by any means, radio, electronic, mechanical, photocopying, recording or facsimile, or otherwise, without the prior written permission of Motorola, Inc.

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

Restricted Rights Legend

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

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

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

Preface

This manual, *MVME147 MPU VMEmodule Installation and Use,* provides general information, hardware preparation and installation instructions, operating instructions, programming information, functional description, and debugger firmware information for the MVME147 MPU VMEmodule. The information contained in this manual applies to the following MVME147 models:

MVME147-010
MVME147-011
MVME147-012
MVME147-013
MVME147-014
MVME147-022
MVME147-023
MVME147-024

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

A basic knowledge of computers and digital logic is assumed.

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

The computer programs stored in the Read Only Memory of this device contain material copyrighted by Motorola Inc., 1995, and may be used only under a license such as those contained in Motorola's software licenses.

The software described herein and the documentation appearing herein are furnished under a license agreement and may be used and/or disclosed only in accordance with the terms of the agreement.

The software and documentation are copyrighted materials. Making unauthorized copies is prohibited by law. No part of the software or documentation may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means without the prior written permission of Motorola, Inc.

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 electromagnetic energy. It may cause or be susceptible to electromagnetic interference (EMI) if not installed and used in a cabinet with adequate EMI protection.**

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.

Motorola® and the Motorola symbol are registered trademarks of Motorola, Inc. All other products mentioned in this document are trademarks or registered trademarks of their respective holders.

© Copyright Motorola, Inc. 1996

All Rights Reserved

Printed in the United States of America

May 1996

Safety Summary
Safety Depends On You

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

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

Ground the Instrument.

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

Do Not Operate in an Explosive Atmosphere.

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

Keep Away From Live Circuits.

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

Do Not Service or Adjust Alone.

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

Use Caution When Exposing or Handling the CRT.

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

Do Not Substitute Parts or Modify Equipment.

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

Dangerous Procedure Warnings.

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

WARNING

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

List of Tables

General Information 1

Introduction

This manual provides:

- General information
- Hardware preparation and installation instructions
- Operating instructions
- Functional description

for the MVME147 series of MPU VMEmodules (referred to as the MVME147 throughout this manual).

Model Designations

The MVME147 is available in several models which are listed in Table 1-1. The memory maps differ for the 4, 8, 16, and 32MB versions (refer to the *Memory Map*s section in Chapter 3).

Table 1-1. MVME147 Model Designations

Model Number	Clock Speed	Memory	Parity	Ethernet
MVME147-010	16 MHz	4MB	No	No
MVME147-011	25 MHz	4MB	Yes	Yes
MVME147-012	25 MHz	8MB	Yes	Yes
MVME147-013	25 MHz	16MB	Yes	Yes
MVME147-014	25 MHz	32MB	Yes	Yes
MVME147-022	32 MHz	8MB	Yes	Yes
MVME147-023	32 MHz	16MB	Yes	Yes
MVME147-024	32 MHz	32MB	Yes	Yes

Features

Features of the MVME147 are listed in Table 1-2:

Table 1-2. MVME147 Features

Feature	Description
Microprocessor	MC68030
Floating-point coprocessor	MC68882
DRAM	Shared DRAM with parity (no parity on MVME147-010)
ROM	Four ROM/PROM/EPROM/EEPROM sockets (organized as 16 bits wide)
Status LEDs	Four LEDs: SCON, RUN, FAIL, and STATUS
CMOS RAM	4K by 8 available with battery backup
Switches	Two switches: RESET and ABORT
Remote reset connector	One connector for remote access of RESET switch
Real time clock	TOD clock/calendar with battery backup
Tick timers	Two 16-bit tick timers for periodic interrupts
Watchdog timer	One watchdog timer
Software interrupts	Two software interrupts
I/O	SCSI bus interface with DMA
	Four serial ports EIA-232-D interface
	8-bit Centronics parallel printer port
	Ethernet transceiver interface (except for MVME147-010)
VMEbus interface	VMEbus system controller functions
	VMEbus master interface (A32/D32, A24/D16 compatible)
	VMEbus interrupter
	VMEbus requester

Specifications

General specifications for the MVME147 are listed in Table 1-3.

The following sections detail cooling requirements and FCC compliance.

Cooling Requirements

Motorola VMEmodules are specified, designed, and tested to operate reliably with an incoming air temperature range from 0 degrees C to 55 degrees C (32 degrees F to 131 degrees F) with forced air cooling. Temperature qualification is performed in a standard Motorola VMEsystem- 1000 chassis. Twenty-five watt load boards are inserted in the 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 MVME card cage. The incoming air temperature is measured between the fan assembly and the card cage where the incoming airstream first encounters the module under test. Test software is executed as the module is subjected to ambient temperature variations. Case temperatures of critical, high power density integrated circuits are monitored to ensure component vendors specifications are not exceeded.

While the exact amount of airflow required for cooling depends on 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 flowing over the module. Less air flow is required to cool the module in environments having lower maximum ambients. Under more favorable thermal conditions it may be possible to operate the module reliably at higher than 55 degrees C with increased air flow. 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 of air flowing over a module.

FCC Compliance

The MVME147 was tested in an FCC-compliant chassis, and meets the requirements for Class A equipment. FCC compliance was achieved under the following conditions:

1. Shielded cables on all external I/O ports.
2. Cable shields connected to earth ground via metal shell connectors bonded to a conductive module front panel.

3. Conductive chassis rails connected to earth ground. This provides the path for connecting shields to earth ground.
4. Front panel screws properly tightened.

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

Table 1-3. MVME147 Specifications

Characteristics	Specifications
Power requirements (MVME147 with two EPROMs and MVME712M)	+5 Vdc, 4.5A maximum (3.5 A typical) +12 Vdc, 1.0 A maximum (100 mA maximum - no LAN) -12 Vdc, 100 mA maximum **Note:** Power must be brought in from both the P1 and P2 backplanes or connectors P1 and P2
Microprocessor	MC68030
Clock signal	16/25/32 MHz to MPU and FPC (depends on version)
Addressing	
Total address range (on and off board)	4GB
EPROM/EEPROM	Four sockets, 32 pin, for 8K x 8 to 1M x 8 devices
Dynamic RAM	4/8/16/32 MB (depends on version)
I/O ports	
Serial	Four multiprotocol serial ports (connected through P2 to transition module)
Parallel	Parallel I/O Centronics printer port (connected through P2 to transition module)
Timers	Four total
Time-of-day clock	M48T18 (only 4KB SRAM accessible)
Watchdog timer	16-bit (tick timer output is watchdog timer input)
Tick timer	Two 16-bit programmable
Bus configuration	Data transfer bus master, with 32-bit address (A32) and 32-bit data (D32) (A24:D16 also supported)
Interrupt handler	Any or all onboard, plus up to seven VMEbus interrupts
Bus arbitration	Two modes: prioritized mode and rotating priority mode

Table 1-3. MVME147 Specifications (Continued)

Characteristics	Specifications
Reset	RESET switch which can be enabled/disabled by software. If the MVME147 is the system controller, it also activates SYSRESET* (system reset) on the VMEbus
Operating temperature	0 degrees to 55 degrees C at point of entry of forced air (approximately 490 LFM)
Storage temperature	-40° to 85° C
Relative humidity	-5% to 90% (non-condensing)
Physical characteristics (excluding front panel)	
Height	9.187 inches (233.35 mm)
Depth	6.299 inches (160.0 mm)
Thickness	0.063 inches (1.6 mm)

General Description

The MVME147 is a double-high VMEmodule based on the MC68030 microprocessor. It is best utilized in a 32-bit VMEbus system with both P1 and P2 backplanes. The module has high functionality with large onboard shared RAM, serial ports, and Centronics printer port. The module provides a SCSI bus controller with DMA, floating-point coprocessor, tick timer, watchdog timer, and time-of-day clock/calendar with battery backup, 4KB of static RAM with battery backup, four ROM sockets, and A32/D32 VMEbus interface with system controller functions.

The MVME147 can be operated as part of a VMEbus system with other VMEmodules such as RAM modules, CPU modules, graphics modules, and analog I/O modules. The following transition boards are compatible with the MVME147:

- ❏ MVME712-12
- ❏ MVME712-13
- ❏ MVME712A
- ❏ MVME712AM
- ❏ MVME712B

❏ MVME712M

Equipment Required

The following equipment is required to make a complete system using the MVME147:

- ❑ Terminal
- ❑ Disk drives and controllers
- ❑ Transition module(s) and connecting cables:
  - MVME712-12
  - MVME712-13
  - MVME712A
  - MVME712AM
  - MVME712B
  - MVME712M

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

Note The MVME712B is designed to be used only in conjunction with an MVME712-12/-13/A/AM for external SCSI and/or Ethernet connections.

The MVME147Bug debug monitor firmware (147Bug) is provided in the two EPROMs in sockets on the MVME147. It provides:

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

147Bug includes a user interface which accepts commands from the system console terminal. 147Bug can also operate in a System Mode, which includes choices from a service menu. Refer to Appendix B, *Debugger General Information*.

The MVME712 modules provide the interface between the MVME147 and peripheral devices. They connect the MVME147 to EIA-232-D serial devices, Centronics-compatible parallel devices, SCSI devices, and Ethernet devices. The MVME712 is cabled to the MVME147 through the P2 adapter board.

The features of the MVME712 modules are shown in Table 1-4. They include:

- ❏ Four multiprotocol EIA-232-D serial ports
- ❏ One independent printer port
- ❏ Small computer systems interface (SCSI) shielded connector bus interface for connection to external devices
- ❏ Ethernet interface
- ❏ Built-in modem with front panel Telco modular jack (MVME712-13 and MVME712AM)
- ❏ Electrostatic discharge (ESD) protection on front panel
- ❏ Radio frequency interference (RFI) protection on front panel

Table 1-4. MVME712 Transition Modules

Board	Panel Size	EIA-232-D Ports	Serial Connector	Printer	Ethernet	SCSI	Modem
MVME712M	Double	4	25 pin	Yes	Yes	Yes	No
MVME712A	Single	4	9 pin	Yes	*	*	No
MVME712AM	Single	4	9 pin	Yes	*	*	Yes
MVME712-12	Single	4	9 pin	Yes	*	*	No
MVME712-13	Single	4	9 pin	Yes	*	*	Yes
MVME712B	Single	0	No	No	Yes	Yes	No
* These functions can be supplied by using the MVME712B							

Related Documentation

The following publications are applicable to the MVME147 and may provide additional helpful information. If not shipped with this product, they may be purchased by contacting your local Motorola sales office. Non-Motorola documents may be purchased from the sources listed.

Document Title	Motorola Publication Number[1]
MVME147 MPU VMEmodule Installation and Use [2] (this manual)	VME147A/IH
MVME147BUG 147Bug Debugging Package User's Manual, Parts 1 and 2[2]	V147BUGA1/UM V147BUGA2/UM
MVME147 SCSI Firmware User's Manual[2]	MVME147FW/D
MC68881/MC68882 Floating-Point Coprocessor User's Manual	MC68881UM/AD
MC68030 Enhanced 32-Bit Microprocessor User's Manual	MC68030UM/AD
M68000-/08-/16-/32- Bit Microprocessor User’s Manual	M68000UM/AD
M68000 Family Reference Manual	M68000FR/AD
MVME712-12, MVME712-13, MVME712A, MVME712AM, and MVME712B Transition Modules and LCP2 Adapter Board User's Manual	MVME712A/D
MVME712M Transition Module and P2 Adapter Board User's Manual	MVME712M/D

Notes 1. Although not shown in the above list, each Motorola Computer Group manual publication number is suffixed with characters that represent the type and revision level of the document, such as “/xx2” (the second revision of a manual); a supplement bears the same number as a manual but has a suffix such as “/xx2A1” (the first supplement to the second revision of the manual).

2. Manuals shown with a superscript ([2]) can be ordered as a set with the part number LK-147SET.

The following publications are available from the sources indicated:

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

SCC User's Manual; Zilog, Inc., 210 East Hacienda Avenue, Campbell, CA 95008-6600

SCSI Small Computer System Interface; draft X3T9.2/82-2 - Revision 14; Computer and Business Equipment Manufacturers Association, 311 First Street, N. W., Suite 500, Washington, D.C. 20001

M48T18, CMOS 8Kx8 TIMEKEEPER ™ SRAM data sheet in Non-Volatile RAM Products Databook; SGS-THOMSON Microelectronics, Inc., 1000 East Bell Road, Phoenix, AZ 85022-2699.

WD33C93 SCSI-Bus Interface Controller; Western Digital Corporation, 2445 McCabe Way, Irvine, CA 92714.

Local Area Network Controller Am79C90 (LANCE), Technical Manual, order number 06363A, Advanced Micro Devices, Inc., 901 Thompson Place, P.O Box 3453, Sunnyvale, CA 94088.

Support Information

You can obtain parts lists and schematics for the MVME147 by contacting your local Motorola sales office.

Manual Terminology

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

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

Unless otherwise specified, all address references are in hexadecimal.

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

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

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

Data and address sizes are defined as follows:

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

Hardware Preparation and Installation 2

Introduction

This chapter provides the following for the MVME147:

- ❑ Unpacking instructions
- ❑ Hardware preparation
- ❑ Installation instructions

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

Unpacking Instructions

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

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

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

Overview of Start-up Procedure

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

Table 2-1. Start-up Overview

What you will need to do ...	Refer to ...	On page ...
Set jumpers on your MVME147 module.	*Hardware Preparation*	2-4
Ensure that ROM devices are properly installed in the sockets.	*Hardware Preparation*	2-4
Install your MVME147 module in the chassis.	*Installation Instructions*	2-14
Set jumpers on the transition board; connect and install the transition board, P2 adapter module, and optional SCSI device cables.	The user's manual you received with your MVME712 module, listed in *Related Documentation*	1-9
	You may also wish to obtain the *MVME147 SCSI Firmware User's Manual*, listed in *Related Documentation*	1-9
Connect a console terminal to the MVME712.	*Installation Instructions*	2-14
	The user's manual you received with your MVME712 module, listed in *Related Documentation*	1-9
Connect any other optional devices or equipment you will be using.	The user's manual you received with your MVME712 module, listed in *Related Documentation*	1-9
	EIA-232-D Interconnections	A-1
	Port Numbers	B-30
Power up the system.	*Installation Instructions*	2-14
	Front Panel Indicators (DS1 - DS4)	3-1
	Troubleshooting; Solving Start-up Problems	D-1

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

What you will need to do ...	Refer to ...	On page ...
Note that the debugger prompt appears.	*Installation Instructions*	2-14
	Debugger General Information.	B-1
	You may also wish to obtain the *MVME147BUG - 147Bug Debugging Package User's Manual*, listed in *Related Documentation*	1-9
Initialize the clock.	*Installation Instructions*	2-14
	SET and ENV Commands	C-1
Examine and/or change environmental parameters.	*Installation Instructions*	2-14
	SET and ENV Commands	C-1
Program the PCCchip and VMEchip.	*Memory Maps*	3-4
	Programming	4-1

Hardware Preparation

The MVME147 has been factory tested and is shipped with factory-installed jumpers configured to provide the system functions required for a VMEbus system. The module is operational with the factory-installed jumpers, but to select the desired configuration and ensure proper operation of the MVME147, certain option modifications may be necessary.

Options are selected by installing or removing jumpers on the following headers:

- ❑ ROM configuration select (J1, J2)
- ❑ System controller select (J3)
- ❑ Serial port 4 clock configuration select (J8, J9)

Instructions for setting the jumpers are given in the following pages. Refer to Figure 2-1 for the location of:

- ❑ Headers
- ❑ Connectors
- ❑ LEDs
- ❑ Switches

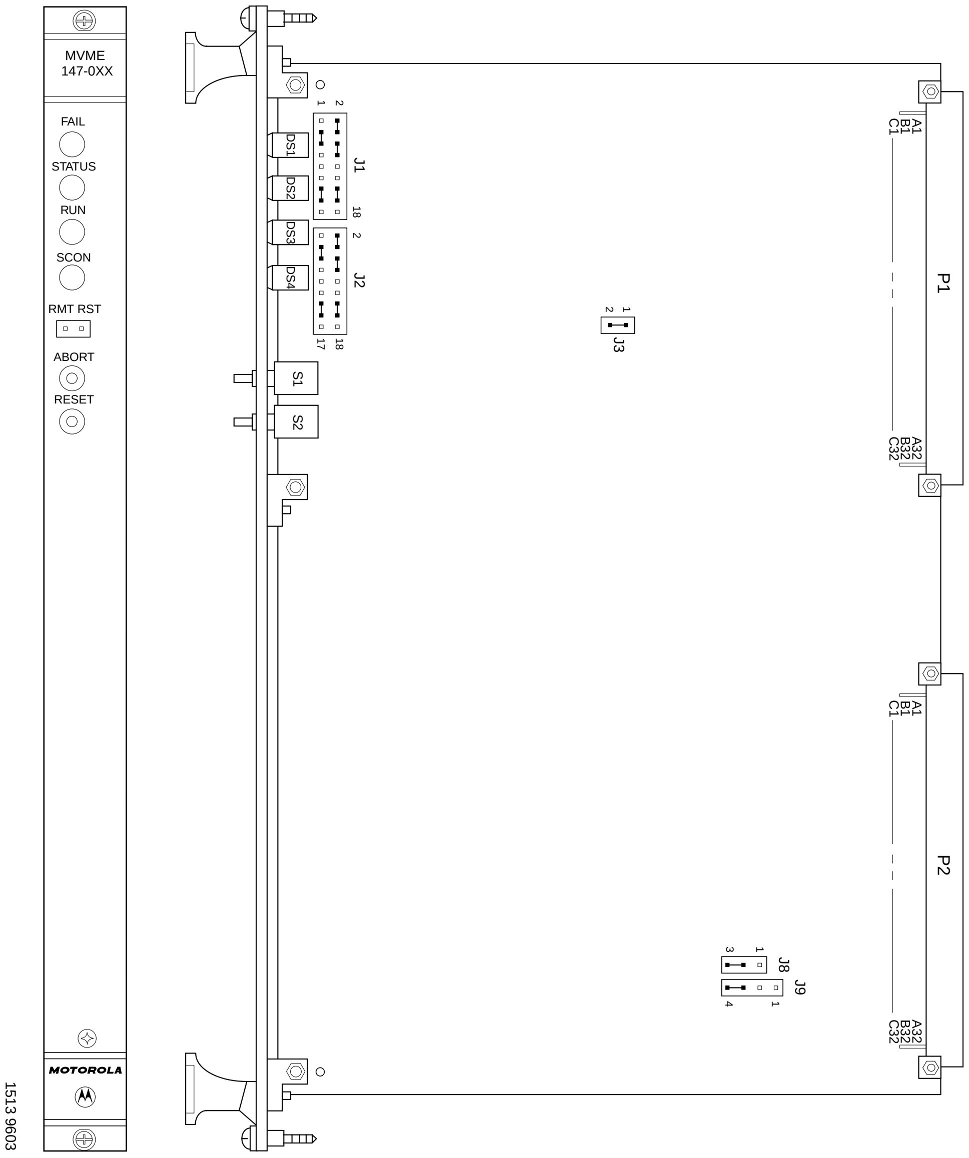


Figure 2-1. MVME147 Header Locations

ROM Configuration Select Headers (J1, J2)

The MVME147 supports various sizes of EPROMs and EEPROMs.

Four 32-pin ROM/PROM/EPROM/EEPROM sockets are provided on the module. They are organized as two banks with two sockets per bank. Each pair of sockets may be individually configured. Sockets U22 and U30 form bank 1. Sockets U1 and U15 form bank 2.

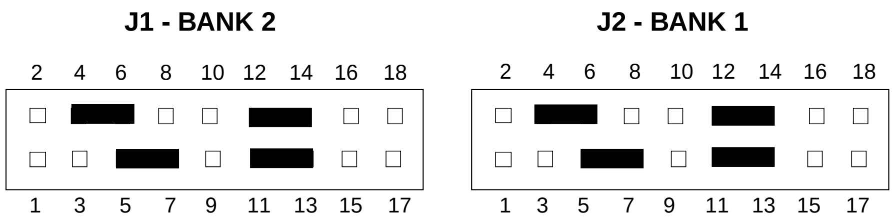
The banks are configured as word ports to the MPU with U22 and U1 comprising the even bytes and U30 and U15 the odd bytes. Each bank can be configured for 8K x 8, 16K x 8, 32K x 8, 64K x 8, 128K x 8, 256K x 8, 512K x 8, or 1M x 8 ROM/PROM/EPROM devices; or for 2K x 8, 8K x 8, or 32K x 8 EEPROM devices.

As shipped, the module is configured for 128K x 8 ROM/PROM/EPROM devices (Configuration #5), and the MVME147Bug debugger software EPROMs are installed in sockets U22 and U30.

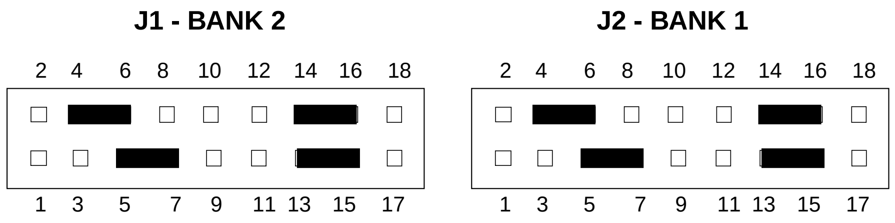
Note There are several different algorithms for erasing/writing to EEPROM devices depending on the manufacturer. The MVME147 supports only those devices which have a “static RAM” compatible erase/write mechanism such as Xicor X28256 or X2864H.

J1 and J2 Jumpers

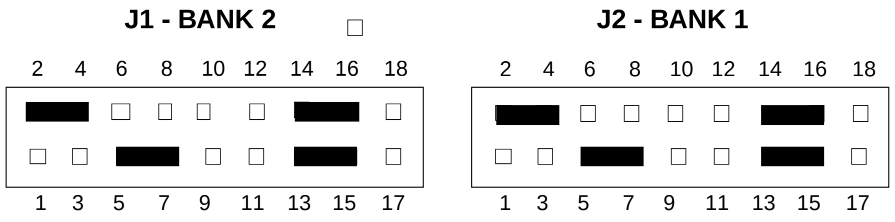
The J1 and J2 headers on the MVME147 module must be configured for the ROM device type used, as shown on the following pages.



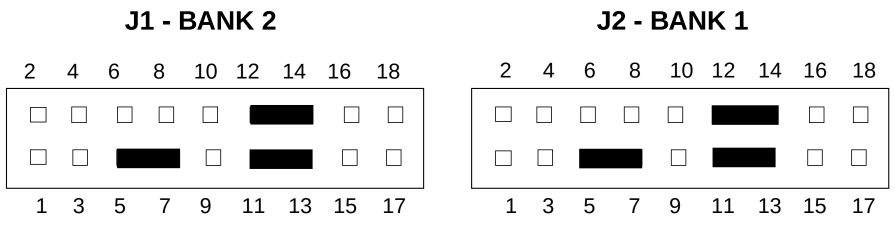
Configuration #1: 8K x 8 or 16K x 8 ROM/PROM/EPROM



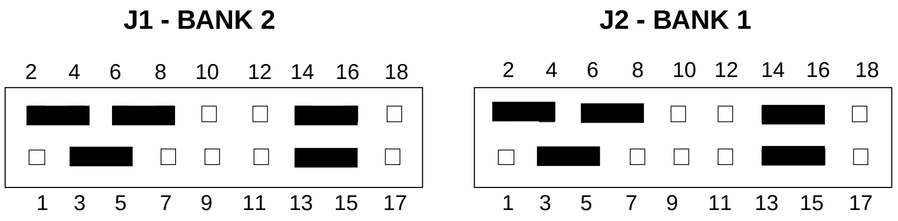
Configuration #2: 32K x 8 ROM/PROM/EPROM



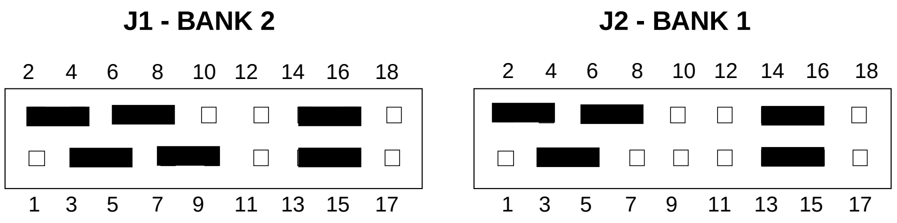
Configuration #3: 64K x 8 ROM/PROM/EPROM



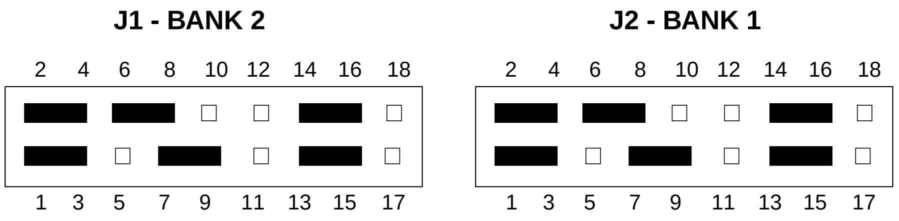
Configuration #4: 2K x 8 or 8K x 8 EEPROM



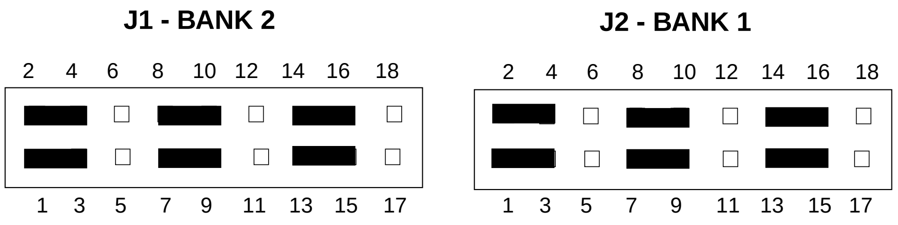
Configuration #5: 128K x 8 ROM/PROM/EPROM
(Factory Configuration)



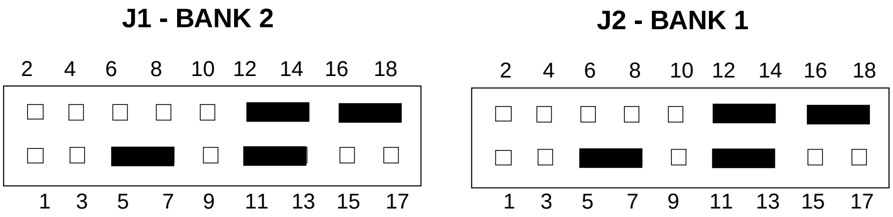
Configuration #6: 256K x 8 ROM/PROM/EPROM/EPROM



Configuration #7: 512K x 8 ROM/PROM/EPROM



Configuration #8: 1M x 8 ROM/PROM/EPROM



Configuration #9: 32K x 8 EEPROM

Socket Pin Definitions

The sockets are installed on the module with pins oriented as shown in Figure 2-2. Devices with 28 pins are installed with pin 1 of the device aligned with pin 3 of the socket as shown.

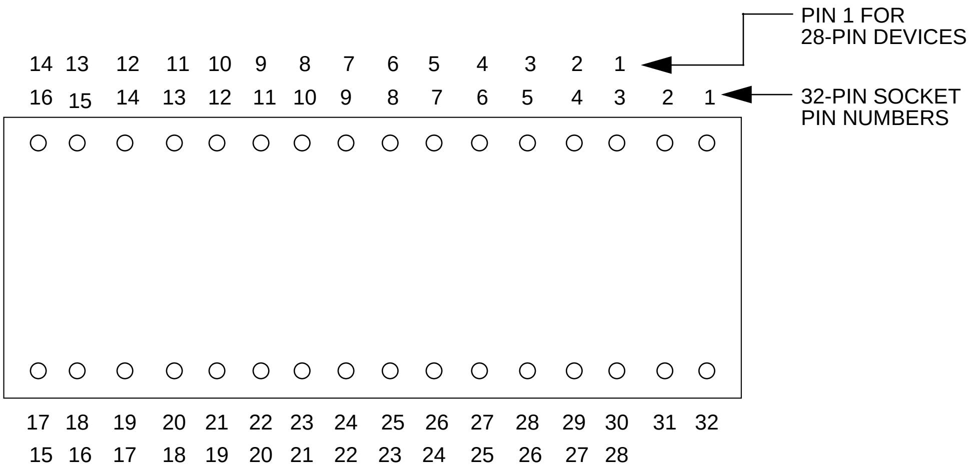


Figure 2-2. Socket Alignment

Figure 2-3 shows the definitions of the ROM/PROM/EPROM/EEPROM socket pins, depending upon the configuration used. The address lines shown are local bus address lines, not device address lines.

The configurations shown in the figure are as follows:

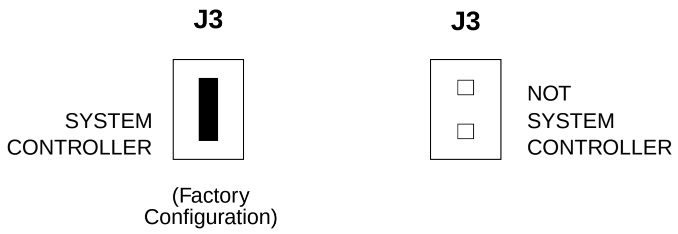
Configuration Number	Device Type
1	8K x 8, 16K x 8 EPROM
2	32K x 8 EPROM
3	64K x 8 EPROM
4	2K x 8 (28-pin), 8K x 8 EEPROM
5	128K x 8 EPROM
6	256K x 8 EPROM
7	512K x 8 EPROM
8	1M x 8 EPROM
9	32K x 8 EEPROM

Configuration													Configuration								
1	2	3	4	5	6	7	8	9					9	8	7	6	5	4	3	2	1
				+5V	+5V	+5V	A20		1			32		+5V	+5V	+5V	+5V	+5V	+5V	+5V	+5V
17	A17	A17	A17	A17	A17	A17	A17		2			31		A19	A19	+5V	+5V				
5V	+5V	A16		A16	A16	A16	A16	A15	3	1	28	30	+5V	A18	A18	A18	A18	+5V	+5V	+5V	+5V
13	A13	A13	A13	A13	A13	A13	A13	A13	4	2	27	29	WE*	A15	A15	A15	A15	WE*	A15	A15	WE*
8	A8	A8	A8	A8	A8	A8	A8	A8	5	3	26	28	A14	A14	A14	A14	A14	A14	A14	A14	A14
7	A7	A7	A7	A7	A7	A7	A7	A7	6	4	25	27	A9	A9	A9	A9	A9	A9	A9	A9	A9
6	A6	A6	A6	A6	A6	A6	A6	A6	7	5	24	26	A10	A10	A10	A10	A10	A10	A10	A10	A10
5	A5	A5	A5	A5	A5	A5	A5	A5	8	6	23	25	A12	A12	A12	A12	A12	A12	A12	A12	A12
4	A4	A4	A4	A4	A4	A4	A4	A4	9	7	22	24	OE*	OE*	OE*	OE*	OE*	OE*	OE*	OE*	OE*
3	A3	A3	A3	A3	A3	A3	A3	A3	10	8	21	23	A11	A11	A11	A11	A11	A11	A11	A11	A11
2	A2	A2	A2	A2	A2	A2	A2	A2	11	9	20	22	CE*	CE*	CE*	CE*	CE*	CE*	CE*	CE*	CE*
1	A1	A1	A1	A1	A1	A1	A1	A1	12	10	19	21	D7	D7	D7	D7	D7	D7	D7	D7	D7
0	D0	D0	D0	D0	D0	D0	D0	D0	13	11	18	20	D6	D6	D6	D6	D6	D6	D6	D6	D6
1	D1	D1	D1	D1	D1	D1	D1	D1	14	12	17	19	D5	D5	D5	D5	D5	D5	D5	D5	D5
2	D2	D2	D2	D2	D2	D2	D2	D2	15	13	16	18	D4	D4	D4	D4	D4	D4	D4	D4	D4
nd	Gnd	Gnd	Gnd	Gnd	Gnd	Gnd	Gnd	Gnd	16	14	15	17	D3	D3	D3	D3	D3	D3	D3	D3	D3

Figure 2-3. Socket Pin Definitions

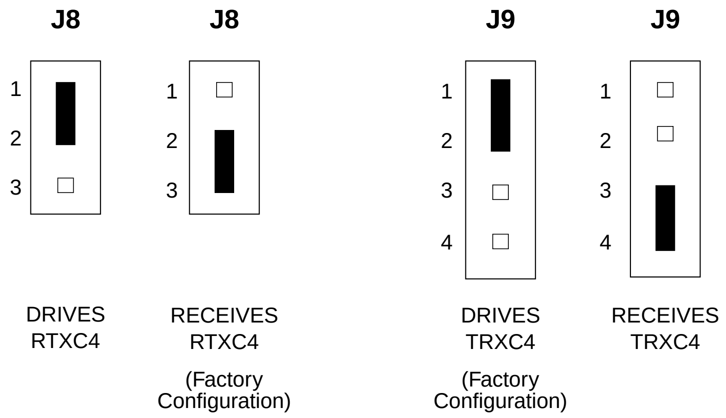
System Controller Select Header (J3)

Header J3 allows you to select the MVME147 as system controller. With the jumper removed, the module is not used as system controller. The module is shipped with the jumper installed (system controller).



Serial Port 4 Clock Configuration Select Headers (J8, J9)

Serial port 4 can be configured to use clock signals provided by the TRXC4 and RTXC4 signal lines. Headers J8 and J9 on the MVME147 module configure part of the clock signals. The remaining configuration of the clock lines is accomplished using header J15 on the MVME712M module. Refer to the *MVME712M Transition Module and P2 Adapter Board User's Manual* for header J15 configuration. The module is shipped with the jumper on pins 2 and 3 of J8 (receiving RTXC4), and on pins 1 and 2 of J9 (driving TRXC4).



Installation Instructions

When you have configured the MVME147's headers and installed the selected ROMs in the sockets as described previously, install the MVME147 module in the system as follows:

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

Caution Connecting modules while power is applied may result in damage to components on the module.

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 equipment user's manual.
3. Remove the filler panel(s) from the appropriate card slot(s) at the front and rear of the chassis (if the chassis has a rear card cage). If the MVME147 is configured as the system controller, install it in the left-most card slot (slot 1) to initiate the bus grant daisy-chain correctly. The MVME147 is to be installed in the front of the chassis; the MVME712 transition module may be installed in the front or rear of the chassis.

Note Every MVME147 is assigned an Ethernet station address. The address is $08003E2*xxxxx* where *xxxxx* is the unique number assigned to the module; i.e., every MVME147 has a different value for *xxxxx*.

Each Ethernet station address is displayed on a label attached to the MVME147's backplane connector, P2. In addition, the *xxxxx* portion of the Ethernet station address is stored in BBRAM location $FFFE0778 as $2*xxxxx*.

If Motorola networking software is running on an MVME147, it uses the 2*xxxxx* value from BBRAM to complete the Ethernet station address ($08003E2*xxxxx*). You must ensure that the value of 2*xxxxx* is maintained in BBRAM. If the value of 2*xxxxx* is lost in BBRAM, use the number on the backplane connector P2 label to restore it. Note that MVME147Bug includes the "LSAD" command for examining and updating the BBRAM *xxxxx* value.

If non-Motorola networking software is running on an MVME147, it must set up the 79C90 so that the Ethernet station address is that shown on the front panel label to ensure that the module has a globally unique Ethernet station address.

4. Insert the MVME147 into the selected card slot. Be sure the module is seated properly into the connectors on the backplane. Fasten the module in the chassis with the screws provided. For proper operation, a 32-bit VMEbus backplane should be used. This ensures that power is sufficiently distributed over enough power pins on connectors P1 and P2.

5. Remove IACK and BG jumpers from the header on chassis backplane for the card slot in which the MVME147 is installed (if applicable).

6. Refer to the MVME712 user's manual provided with your MVME712 transition module, and install the transition board, P2 adapter board, and any peripheral cables needed, according to the installation instructions given there.

7. Connect the terminal which is to be used as the system console to the serial I/O port marked **SERIAL PORT 1/CONSOLE** on the MVME712 module. To use 147Bug, set up the terminal as follows:

   – Eight bits per character
   – One stop bit per character
   – Parity disabled (no parity)
   – 9600 baud to agree with the MVME147 ports' default baud rate at power-up.

After power-up, the baud rate of **PORT 1** can be reconfigured by using the Port Format (**PF**) command of the 147Bug debugger.

Note In order for high-baud rate serial communication between 147Bug and the terminal to work, the terminal must do some handshaking. If the terminal being used does not do hardware handshaking via the CTS line, then it must do XON/XOFF handshaking. If you get garbled messages and missing characters, then you should check the terminal to make sure XON/XOFF handshaking is enabled.

8. If you want to connect device(s) (such as a host computer system or a serial printer) to the serial ports on your MVME712/MVME712M, connect the appropriate cables as described in the MVME712 user's manual, and configure the port(s) as described in the section *Port Numbers* in Appendix B of this manual, *Debugger General Information*. After power-up, these ports can be reconfigured by using the **PF** command of the 147Bug debugger.
9. Turn equipment power ON.

Notes The MVME147 provides +12 Vdc power to the Ethernet transceiver interface through a 1-amp polyfuse located between the P1 and P2 connectors on the MVME147 module. The polyfuse resets itself when the overcurrent condition no longer exists. The polyfuse is soldered down and should not need to be replaced. When using an MVME712M module, the yellow LED (DS1) on the MVME712M front panel will light when LAN power is available. Note that the yellow LED may light if a device is connected to any one of the four serial ports on the MVME712M regardless of the state of the fuse.

The MVME147 provides SCSI terminator power through a 1-amp fuse (F1) located on the P2 Adapter Board. The fuse is socketed. If the fuse is blown, the

SCSI devices may not operate or may function erratically. When the P2 adapter is used with an MVME712M and the SCSI bus is connected to the MVME712M, the green LED (DS2) on the MVME712M lights when there is SCSI terminator power. If the LED flickers during SCSI bus operations, the fuse should be checked.

10. Observe the system console. The default condition is with the ROMboot routine enabled, so the boot routine contained in the MVME147's ROMs is begun.

    With the 147Bug EPROMs installed on the MVME147 module, 147Bug executes a set of self-tests and displays its prompt, `147Bug>`. The **RUN** LED on the MVME147 module will be lit.

    If after a delay, 147Bug begins to display test result messages on the bottom line of the screen in rapid succession, the MVME147 is in 147Bug's "System" operating mode. To get to the normal or "Bug" operating mode, press the **ABORT** switch on the front panel of the MVME147 to cause a menu to be displayed. Enter a **3** to go to the system debugger. (Refer to Appendix B.)

    When power is applied to the MVME147, bit 1 at location $FFFE1029 (Peripheral Channel Controller (PCC) general purpose status register) is set to 1 indicating that power was just applied. (Refer to Chapter 4 for a description of the PCC.) This bit is tested within the "Reset" logic path to see if the power-up confidence test needs to be executed. This bit is cleared by writing a 1 to it thus preventing any future power-up confidence test execution.

    If the power-up confidence test is successful and no failures are detected, the firmware monitor comes up normally, with the **FAIL** LED off.

    If the confidence test fails, the test is aborted when the first fault is encountered and the **FAIL** LED remains on. If possible, one of the following messages is displayed and the firmware monitor comes up with the **FAIL** LED on:

```
... 'CPU Register test failed'
... 'CPU Instruction test failed'
... 'ROM test failed'
... 'RAM test failed'
... 'CPU Addressing Modes test failed'
... 'Exception Processing test failed'
... '+12v fuse is open'
... 'Battery low (data may be corrupted)'
... 'Unable to access non-volatile RAM properly'
```

Refer to Appendix D, *Troubleshooting: Solving Start-up Problems*.

11. At the 147Bug prompt, use the **SET** command to initialize the clock and set the time and date. Refer to Appendix C, *SET and ENV Commands.*

Caution

Performing the next step will change some parameters that may affect your system operation.

12. Use 147Bug's **ENV** command to verify and/or change the environmental parameters in NVRAM. The **ENV;D** command can be used to return the NVRAM parameters to the default values. Refer to Appendix C, *SET and ENV Commands*.

    If you wish to use 147Bug's Autoboot routine in order to boot automatically from another controller and device, use the **AB** command (Appendix B).

To program the MVME147 module's PCCchip and VMEchip, refer to the memory maps in Chapter 3 and to Chapter 4, *Programming.*

Operating Instructions 3

Introduction

This chapter provides information on using the MVME147 in a system configuration. The following topics are presented:

- ❑ Controls and Indicators, page 3-1
- ❑ Memory Maps, page 3-4

Controls and Indicators

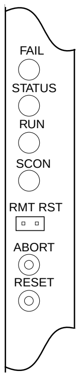
There are two switches on the front panel of the MVME147:

- ❑ **ABORT**
- ❑ **RESET**

There are four LED status indicators on the MVME147 front panel:

- ❑ **FAIL**
- ❑ **STATUS**
- ❑ **RUN**
- ❑ **SCON**

Each is described below and the indicators are summarized in Table 3-1.



ABORT Switch (S1)

The front panel software **ABORT** switch is normally used to abort program execution and return to the debugger.

The Peripheral Channel Controller (PCC) provides the **ABORT** switch interface. The **ABORT** switch signal is debounced and sent to the level-7 interrupter. When it is enabled, the **ABORT** causes a level-7 interrupt to the MC68030. The interrupter returns a status/ID vector when requested.

The **ABORT** switch is enabled when using 147Bug. It is enabled/ disabled by software.

RESET Switch (S2)

The front panel **RESET** switch S2 generates a local reset, when enabled. It also generates a VMEbus System Reset, if the MVME147 is the system controller.

If the MVME147 is not the system controller, this switch should not be used when the local MPU is executing VMEbus cycles.

The PCC provides the **RESET** switch interface. The **RESET** switch signal is debounced and when it is enabled, it causes a reset out signal.

The **RESET** switch is enabled at power-up, and 147Bug leaves it enabled. It is enabled/disabled by software.

RMT RST Switch Connector (J4)

The remote reset connector J4, marked **RMT RST**, allows the front panel reset function to be provided remotely by a user-supplied/installed switch/cable assembly. Shorting the two pins has the same effect as pressing the front panel **RESET** switch.

When the remote switch and cable is installed, the MVME147 can be reset by the remote switch or by the front panel **RESET** switch.

FAIL Indicator (DS1)

The red **FAIL** LED, DS1, indicates the status of the BRDFAIL bit in the VMEchip. The **FAIL** LED lights when the BRDFAIL bit is set or when watchdog time-out occurs in the PCC. Also, if the **FAIL** LED is lit and SYSFAIL inhibit bit in the VMEchip is not set, the MVME147 drives SYSFAIL on the VMEbus.

STATUS Indicator (DS2)

The yellow **STATUS** LED, DS2, lights whenever the MC68030 STATUS* pin is low. When the yellow LED is fully lit, the processor has halted.

RUN Indicator (DS3)

The green **RUN** LED, DS3, is connected to the MC68030 address strobe (AS*) signal and indicates that the MPU is executing a bus cycle.

SCON Indicator (DS4)

The green **SCON** LED, DS4, lights when the MVME147 is the VMEbus system controller.

Table 3-1. Front Panel Indicators and MVME147 Status

FAIL DS1 (Red)	STATUS DS2 (Yellow)	RUN DS3 (Green)	MVME147 Status
off	off	off	No power is applied to the module, or the MPU is not the current local bus master.
off	off	ON	MPU is waiting for a cycle to complete.
off	ON (bright)	off	MPU is halted.
off	ON (normal)	off	MPU is executing out of its onchip cache only.
off	ON	ON	Normal operation.
ON	off	off	MPU is not current local bus master and is not executing out of onchip cache. Also, [BRDFAIL] has not been cleared since reset or has been set by software.
ON	off	ON	[BRDFAIL] has not been cleared since reset or has been set by software. Also, MPU is waiting for a cycle to complete.
ON	ON (bright)	off	MPU is halted and [BRDFAIL] has not been cleared since reset or has been set by software.
ON	ON (normal)	off	[BRDFAIL] has not been cleared since reset or has been set by software. Also, MPU is executing out of onchip cache only.
ON	ON	ON	[BRDFAIL] has not been cleared since reset or has been set by software.

Memory Maps

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

- ❑ The mapping of all resources as viewed by the MC68030 (MC68030 memory map)
- ❑ The mapping of onboard resources as viewed by VMEbus masters (VMEbus memory map).

MC68030 Memory Map

The MC68030 memory map is split into different address spaces by the function codes. The MVME147 has different groups of devices that respond depending on the address space as shown in Table 3-2.

Table 3-2. MVME147 Address Spaces

FC (2-0)	Address Space	MVME147 Devices that Respond
0	Reserved	None (causes local time-out)
1	User data	All except interrupt handler and MC68882
2	User program	All except interrupt handler and MC68882
3	Reserved	None (causes local time-out)
4	Reserved	None (causes local time-out)
5	Supervisor data	All except interrupt handler and MC68882
6	Supervisor program	All except interrupt handler and MC68882
7	CPU (IACK)	Interrupt handler
7	CPU (coprocessor)	MC68882

Program and Data Address Spaces

The memory map of devices that respond in user data, user program, supervisor data, and supervisor program spaces is shown in the following tables. The entire map from $00000000 to $FFFFFFFF is shown in Table 3-3. The I/O devices are further defined in Table 3-4.

Table 3-3. MC68030 Main Memory Map

Address Range	Devices Accessed	Port Size	Size	H/W Cache Inhibit	Notes
00000000-DRAMsize	Onboard DRAM	D32	4-32MB	No	1, 2
DRAMsize-EFFFFFFF	VMEbus A32/A24	D32	3GB	Yes	3, 4
F0000000-F0FFFFFF	VMEbus A24	D16	16MB	Yes	
F1000000-FF7FFFFF	VMEbus A32	D16	232MB	Yes	
FF800000-FF9FFFFF	ROM/EEPROM bank 1	D16	2MB	Yes	
FFA00000-FFBFFFFF	ROM/EEPROM bank 2	D16	2MB	Yes	
FFC00000-FFFDFFFF	Reserved	N/A	4MB	Yes	3
FFFE0000-FFFE4FFF	Local I/O devices	D8/D16/D32	20KB	Yes	
FFFE5000-FFFEFFFF	Reserved	N/A	44KB	Yes	
FFFF0000-FFFFFFFF	VMEbus short I/O	D16	64KB	Yes	

Notes

1. Onboard ROM/PROM/EPROM/EEPROM bank 1 for the first 4 cycles after a reset, onboard DRAM thereafter.

2. DRAM size varies from 4MB to 32MB depending on the MVME147 model.

3. Size is approximate.

4. This A24 only applies to VMEbus space that falls below $01000000. VMEbus space below $01000000 only occurs on MVME147 models that have DRAMsize smaller than 16MB.

Table 3-4. Local I/O Devices

Address Range	Devices Accessed	Port Size	Size	Notes
FFFE0000-FFFE07F7	BB RAM	D8	2040 bytes	
FFFE07F8-FFFE07FF	BB TOD clock	D8	8 bytes	3
FFFE0800-FFFE0FFF	BB RAM (additional)	D8	2048 bytes	
FFFE1000-FFFE100F	PCC 32-bit registers	D32	16 bytes	
FFFE1010-FFFE102F	PCC 16-bit registers	D16	32 bytes	
FFFE1030-FFFE17FF	PCC registers (repeated)	D32/D16		
FFFE1800-FFFE1803	LANCE (AM7990)	D16	4 bytes	3, 4
FFFE1804-FFFE1FFF	LANCE (repeated)	D16		
FFFE2000-FFFE201F	VMEchip registers	D16	32 bytes	
FFFE2020-FFFE27FF	VMEchip registers (repeated)	D16		
FFFE2800	Printer data (write only)	D8	1 byte	
FFFE2800	Printer status (read only)	D8	1 byte	
FFFE2801-FFFE2FFF	Printer registers (repeated)	D8		
FFFE3000-FFFE3001	Serial 2	D8	2 bytes	1, 3
FFFE3002-FFFE3003	Serial 1	D8	2 bytes	1, 3
FFFE3004-FFFE37FF	Serial 2, 1 (repeated)	D8		
FFFE3800-FFFE3801	Serial 4	D8	2 bytes	2, 3
FFFE3802-FFFE3803	Serial 3	D8	2 bytes	2, 3
FFFE3804-FFFE3FFF	Serial 4, 3 (repeated)	D8		
FFFE4000-FFFE401F	SCSI registers (WD33C93)	D8	32 bytes	3, 5
FFFE4020-FFFE4FFF	SCSI registers (repeated)	D8		5

Notes

1. Serial ports 1 and 2 are sections A and B, respectively, of the first Z8530.
2. Serial ports 3 and 4 are sections A and B, respectively, of the second Z8530.
3. For a complete description of the register bits, refer to the data sheet for the specific chip.
4. The LAN chip is not installed on the MVME147-010. Access to these addresses results in a local bus time-out.
5. The WD33C93 is interfaced in non-multiplexed mode. Only addresses $FFFE4000 (address/status register) and $FFFE4001 (data register) are necessary for operation. All accesses to the WD33C93 go through the PCC.

CPU Address Space

The MVME147 responds to two types of CPU space cycles:

- ❏ Coprocessor
- ❏ Interrupt acknowledge

The MC68030 is capable of generating other types of CPU space cycles (using breakpoint acknowledge, access level control, or MOVES instructions), but the MVME147 has no devices that respond to them.

Coprocessor Register Map

The MC68882 is the only coprocessor on the MVME147. The map decoder selects the MC68882 any time the MPU executes a coprocessor cycle with Cp-ID of %001 (FC2-FC0 =%111 and A19-A13 =%0010001). The MC68882 registers are selected by A4-A0 as shown in Table 3-5.

Table 3-5. MC68882 Register Map

A4-A0 (in Binary)	MC68882 Register	Comments	Port Size
%0000*x*	Response	Read only	D16
%0001*x*	Control	Write only	D16
%0010*x*	Save	Read only	D16
%0011*x*	Restore	Read/write	D16
%0100*x*	Reserved		D16
%0101*x*	Command	Write only	D16
%0110*x*	Reserved		D16
%0111*x*	Condition	Write only	D16
%100*xx*	Operand	Read/write	D32
%1010*x*	Register select	Read only	D16
%1011*x*	Reserved		D16
%110*xx*	Instruction address	Read/write	D32
%111*xx*	Operand address	Read/write	D32

Note

Writes to the MC68882 read-only registers are ignored and reads to write-only registers return all 1s.

Interrupt Acknowledge Map

The MC68030 distinguishes interrupt acknowledge cycles from other CPU space cycles by placing the binary value %1111 on A19-A16. It also specifies the level that is being acknowledged using A03-A01. The interrupt handler selects which device within that level is being acknowledged. Refer to *Interrupt Handler* in Chapter 5.

VMEbus Memory Map

The following paragraphs describe the mapping of MVME147 resources as viewed by VMEbus masters.

The MVME147 onboard DRAM, VMEchip global registers, and VMEbus interrupter respond to accesses by VMEbus masters. No other devices on the MVME147 respond to such accesses.

VMEbus Accesses to MVME147 Onboard DRAM

When a VMEbus master accesses the MVME147 onboard DRAM, it must do so using the address modifier selected by a control register in the VMEchip and the base address selected by a control register in the PCC. Refer to Table 3-6.

Table 3-6. DRAM Address as Viewed from the VMEbus

RBA4	RBA3	RBA2	RBA1	RBA0	Beginning Address	Ending Address	Notes
0	0	0	0	0	$00000000	(1 x DRAMsize)-1	
0	0	0	0	1	1 x DRAMsize	(2 x DRAMsize)-1	1, 2
0	0	0	1	0	2 x DRAMsize	(3 x DRAMsize)-1	1, 2
0	0	0	1	1	3 x DRAMsize	(4 x DRAMsize)-1	1, 2
0	0	1	0	0	4 x DRAMsize	(5 x DRAMsize)-1	1, 2
0	0	1	0	1	5 x DRAMsize	(6 x DRAMsize)-1	1, 2
0	0	1	1	0	6 x DRAMsize	(7 x DRAMsize)-1	1, 2
0	0	1	1	1	7 x DRAMsize	(8 x DRAMsize)-1	1, 2
0	1	0	0	0	8 x DRAMsize	(9 x DRAMsize)-1	1, 2
0	1	0	0	1	9 x DRAMsize	(10 x DRAMsize)-1	1, 2
0	1	0	1	0	10 x DRAMsize	(11 x DRAMsize)-1	1, 2
0	1	0	1	1	11 x DRAMsize	(12 x DRAMsize)-1	1, 2
0	1	1	0	0	12 x DRAMsize	(13 x DRAMsize)-1	1, 2
0	1	1	0	1	13 x DRAMsize	(14 x DRAMsize)-1	1, 2
0	1	1	1	0	14 x DRAMsize	(15 x DRAMsize)-1	1, 2
0	1	1	1	1	15 x DRAMsize	(16 x DRAMsize)-1	1, 2
1	0	0	0	0	16 x DRAMsize	(17 x DRAMsize)-1	1, 2
1	0	0	0	1	17 x DRAMsize	(18 x DRAMsize)-1	1, 2
1	0	0	1	0	18 x DRAMsize	(19 x DRAMsize)-1	1, 2
1	0	0	1	1	19 x DRAMsize	(20 x DRAMsize)-1	1, 2
1	0	1	0	0	20 x DRAMsize	(21 x DRAMsize)-1	1, 2
1	0	1	0	1	21 x DRAMsize	(22 x DRAMsize)-1	1, 2
1	0	1	1	0	22 x DRAMsize	(23 x DRAMsize)-1	1, 2
1	0	1	1	1	23 x DRAMsize	(24 x DRAMsize)-1	1, 2
1	1	0	0	0	24 x DRAMsize	(25 x DRAMsize)-1	1, 2
1	1	0	0	1	25 x DRAMsize	(26 x DRAMsize)-1	1, 2
1	1	0	1	0	26 x DRAMsize	(27 x DRAMsize)-1	1, 2

Table 3-6. DRAM Address as Viewed from the VMEbus (Continued)

RBA4	RBA3	RBA2	RBA1	RBA0	Beginning Address	Ending Address	Notes
1	1	0	1	1	27 x DRAMsize	(28 x DRAMsize)-1	1, 2
1	1	1	0	0	$00000000	(1 x DRAMsize)-1	1, 3, 4
1	1	1	0	1	1 x DRAMsize	(2 x DRAMsize)-1	1, 3, 4

Notes

1. DRAMsize = the size of the DRAM. For example, if the 4MB version is used, then DRAMsize = $400000, and (3 x DRAMsize)-1 = $BFFFFF.
2. When beginning address is less then 16MB, the DRAM responds to standard or extended address modifiers. When beginning address is 16MB or greater, the DRAM responds to extended address modifiers only. Note that bits 4 and 5 in the VMEchip Slave Address Modifier Register further control response to standard and extended address modifiers.
3. This combination pertains only to DRAMsize of 16MB or 32MB.
4. The values shown in the table refer to extended addresses only. In the standard address range the DRAM responds to $000000 through $7FFFFF.

VMEbus Short I/O Memory Map

The VMEchip Global Control and Status Register (GCSR) Set appears at odd addresses in the VMEbus short I/O memory map. A map decoder in the VMEchip monitors the address and the address modifier lines and requests the VMEchip global registers when they are selected. Note that the GCSR can only be accessed in Supervisor Data Space; no User Mode accesses are available.

The VMEchip GCSR base address is selected using a control register (GCSR base address configuration register) in the VMEchip Local Control and Status Register (LCSR) as shown in Table 3-7. A MVME147 may access its own VMEchip GCSR via the VMEbus.

The MVME147 (and the MVME147Bug default) powers up with the GCSR base address programmed with $F. This is intentionally done so that the GCSR set is not mapped on the VMEbus.

Table 3-7. VMEchip GCSR as Viewed from the VMEbus

LCSR Register Bits	Short I/O Address of GCSR
$0	$0000-000F
$1	$0010-001F
$2	$0020-002F
$3	$0030-003F
$4	$0040-004F
$5	$0050-005F
$6	$0060-006F
$7	$0070-007F
$8	$0080-008F
$9	$0090-009F
$A	$00A0-00AF
$B	$00B0-00BF
$C	$00C0-00CF
$D	$00D0-00DF
$E	$00E0-00EF
$F	Does not respond

VMEbus Interrupt Acknowledge Map

The VMEbus distinguishes interrupt acknowledge cycles from other cycles by activating the IACK* signal line. It also specifies the level that is being acknowledged using A03-A01. The VMEchip monitors these lines and after receiving IACKIN*, it responds by asserting IACKOUT* if it was not generating an interrupt at the acknowledged level, or by returning a status/ID vector if it was. The MVME147 may handle a VMEbus interrupt generated by its own VMEchip.

Programming 4

Introduction

This chapter provides the information needed to program the Peripheral Channel Controller (PCC) and the VMEchip.

Programming the Peripheral Channel Controller

These sections contain a description of the PCC internal registers and the bit assignments within each register. All registers may be written or read as bytes. Some restrictions apply to bit set and clear instructions and they should not be used, where indicated. An overall view of the PCC is shown in Table 4-1.

Note In the tables for the 8-bit PCC register definitions that follow, the characters in the bottom line define the operations possible on the register bits, as follows:

R	This bit is a read-only status bit.
R/W	This bit is readable and writable.
C	Writing a 1 to this bit clears it. This bit reads 0.
R/C	This bit is readable. Writing a 1 to this bit clears it.

Table 4-1. PCC Overall View

32-BIT REGISTERS		
Address	**Register**	**Function**
FFFE1000	Table address (bits 1 and 0 are zeros)	DMA
FFFE1004	Data address	DMA
FFFE1008	Link -- 0000 -- DFC2-0 -- Byte count (24 bits)	DMA
FFFE100C	Data holding register	DMA

16-BIT REGISTERS		
Address	**Register**	**Function**
FFFE1010	Timer 1 preload	Timer 1
FFFE1012	Timer 1 count	Timer 1
FFFE1014	Timer 2 preload	Timer 2
FFFE1016	Timer 2 count	Timer 2

8-BIT REGISTERS									
Address	**Register**								**Name**
	Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**	
FFFE1018	IntStat				Enable	IL2	IL1	IL0	Timer 1 Int. Cntrl
FFFE1019	Ovf3	Ovf2	Ovf1	Ovf0		ClrOvf	EnaCnt	Enable	Timer 1 Control
FFFE101A	IntStat				Enable	IL2	IL1	IL0	Timer 2 Int. Cntrl
FFFE101B	Ovf3	Ovf2	Ovf1	Ovf0		ClrOvf	EnaCnt	Enable	Timer 2 Control
FFFE101C	IntStat	ACFail			Enable				AC Fail Int. Cntrl
FFFE101D	WdL3	WdL2	Wd	WdL0	WdTO	WdRst	WdClr	Enable	W'dog Timer Cntrl
FFFE101E	IntStat	FaltInt	ACKInt	ACKPol	Enable	IL2	IL1	IL0	Printer Int. Cntrl
FFFE101F					InPrim	Strobe	StbTim	Mode	Printer Control
FFFE1020	IntStat				Enable	IL2	IL1	IL0	DMA Int. Control
FFFE1021	DONE	8BitEr	TblSizEr	DMABEr	TWBEr	MS/SM*	TW	Enable	DMA Cntrl & Stat.
FFFE1022	IntStat				Enable				Bus Error Int. Cntrl
FFFE1023	Inc4	Inc3	Inc2	Inc1	UU	UM	LM	LL	DMA Status
FFFE1024	IntStat	Abort			Enable				Abort Int. Control
FFFE1025						TblFC2	TblFC1	TblFC0	Tbl. Ad. Func. Cntrl
FFFE1026	IntStat			Int/Ext*	Enable	IL2	IL1	IL0	Serial Prt Int. Cntrl

8-BIT REGISTERS (Continued)									
Address	Register								Name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
FFFE1027	RsDis2	RsDis1	RsDis0	MIntEn	LbToEn	WWPar (Note)	ParEn1 (Note)	ParEn0 (Note)	Gen. Purpose Cntrl
FFFE1028	IntStat				Enable	IL2	IL1	IL0	LAN Int. Cntrl
FFFE1029							PuReset	ParErr	Gen. Purpose Stat.
FFFE102A	IntStat	RstInt	SCSIRst	RstSCSI	Enable	IL2	IL1	IL0	SCSI Prt Int. Cntrl
FFFE102B	LANA25	LANA24	WAITRMC	RBA4	RBA3	RBA2	RBA1	RBA0	Slave Base Addr.
FFFE102C	IntStat				Enable	IL2	IL1	IL0	S/W Int. 1 Control
FFFE102D	IVB 7	IVB6	IVB5	IVB4					Int. Vector Base
FFFE102E	IntStat				Enable	IL2	IL1	IL0	S/W Int. 2 Control
FFFE102F	RevL7	RevL6	RevL5	RevL4	RevL3	RevL2	RevL1	RevL0	Revision Level
FFFE2800	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	Printer Data
FFFE2800	ACK	FAULT	SELECT	PE	BSY		LOW	STAT12	Printer Status

Note These bits are duplicated in the General Control Chip (GCC) and serve their real function there. PCC is only an image now.

Table Address Register

This 32-bit read/write register points to a table of physical addresses and byte counts that are used during DMA transfers when table mode is selected. The table address must be longword aligned because bits 0 and 1 are always zero. If the table address has bit 0 or 1 set, they are truncated and no error is generated. These bits are not affected by reset. Refer to Chapter 5 for details on Table Address.

FFFE1000	Table Address	0	0

Data Address Register

This 32-bit read/write register points to the physical address where data is to be transferred. Data can only be transferred to/from onboard DRAM or VMEbus memory. These bits are not affected by reset.

FFFE1004	Data Address

Byte Count Register

This 32-bit read / write register contains a 24-bit byte counter in bits 0-23, a 3-bit function code in bits 24-26, and a link bit in bit 31. The byte counter contains the number of bytes to be transferred. The function code bits are used when data is transferred. When set in a table entry, the link bit indicates there are more entries in the DMA table. This bit is cleared in the last table entry. The link bit is only used in table mode and is never set by the MC68030. These bits are not affected by reset.

ADDRESS	BIT 31	BIT 30	BIT 29	BIT 28	BIT 27	BIT 26	BIT 25	BIT 24	BITS 23-0
FFFE1008	L	0	0	0	0	DFC2	DFC1	DFC0	Byte Count

Data Holding Register

This read only register holds data passing between the SCSI and local buses. These bits are not affected by reset.

FFFE100C	Data Holding Register

Timer 1 Preload Register

This 16-bit read/write register holds the tick timer preload value. When the counter reaches $FFFF, it is loaded with this value and if interrupts are enabled, an interrupt is generated. When running, the counter is incremented every 6.25 microseconds. The following equation should be used to determine the counter value (*n*) for a periodic interrupt of time *t* where *t* is in seconds.

FFFE1010	Tick 1 preload

$$n = 65536 - \frac{t}{6.25 \times 10^{**}-6}$$

The timer may be programmed to generate interrupts at intervals between 6.25 microseconds and 0.4096 seconds. These bits are not affected by reset.

Timer 1 Counter Register

This 16-bit read register is the output of the tick counter. Reads are not synchronized with counter updates.

FFFE1012	Tick 1 counter

Timer 2 Preload Register

This 16-bit read/write register holds the tick timer preload value. Refer to the *Timer 1 Preload Register* section in this chapter.

FFFE1014	Tick 2 preload

Timer 2 Counter Register

This 16-bit read register is the output of the tick counter. Reads are not synchronized with counter updates.

FFFE1016	Tick 2 counter

Timer 1 Interrupt Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE1018	IntStat				Enable	IL2	IL1	IL0
	R/C				R/W	R/W	R/W	R/W

Note Bit set and clear instructions should not be used on this interrupt control register. Because an interrupt is cleared by writing a 1 to the status bit and the status bit is a 1 to indicate a pending interrupt, the read-modify-write sequence may clear a pending interrupt.

Bits 0-2	These bits program the interrupt level the tick timer generates. Because level 0 does not generate an interrupt, this level is intended for polling software. These bits are cleared by reset.
Bit 3	When this bit is high, the interrupt is enabled. The interrupt is disabled when this bit is low. This bit is cleared by reset.
Bit 7	When this bit is high, a tick timer interrupt is being generated at the level programmed in bits 0-2. This bit is edge sensitive and it is set by a carry out of the tick timer when interrupts are enabled. This bit is cleared when a 1 is written to it or when the interrupt is disabled. When cleared, it remains cleared until the next carry out. This bit is cleared by reset.

Timer 1 Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE1019	Ovf3	Ovf2	Ovf1	Ovf0		ClrOvf	EnaCnt	Enable
	R	R	R	R		C	R/W	R/W

Bit 0	When this bit is low, the timer is disabled and the counter is loaded with the preload value. When the bit is high, the counter is enabled and it starts counting up if the counter enable bit (bit 1) is high. This bit is cleared by reset.
Bit 1	When this bit is low, the counter is stopped. The counter value is not changed when the counter is stopped and started with this bit. When this bit is high, the counter is enabled. This bit is cleared by reset.
Bit 2	The overflow counter is cleared by writing a 1 to this bit.
Bits 4-7	These read only bits are the output of the overflow counter. The overflow counter is incremented each time the tick timer rolls over. These bits are cleared by reset.

Timer 2 Interrupt Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE101A	IntStat				Enable	IL2	IL1	IL0
	R/C				R/W	R/W	R/W	R/W

Note Bit set and clear instructions should not be used on this interrupt control register. Because an interrupt is cleared by writing a 1 to the status bit and the status bit is a 1 to indicate a pending interrupt, the read-modify-write sequence may clear a pending interrupt.

Bits 0-2	These bits program the interrupt level the tick timer generates. Because level 0 does not generate an interrupt, this level is intended for polling software. These bits are cleared by reset.
Bit 3	When this bit is high, the interrupt is enabled. The interrupt is disabled when this bit is low. This bit is cleared by reset.
Bit 7	When this bit is high, a tick timer interrupt is being generated at the level programmed in bits 0-2. This bit is edge sensitive and it is set by a carry out of the tick timer when interrupts are enabled. This bit is cleared when a 1 is written to it or when the interrupt is disabled. When cleared, it remains cleared until the next carry out. This bit is cleared by reset.

Timer 2 Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE101B	Ovf3	Ovf2	Ovf1	Ovf0		ClrOvf	EnaCnt	Enable
	R	R	R	R		C	R/W	R/W

Bit 0	When this bit is low, the timer is disabled and the counter is loaded with the preload value. When the bit is high, the counter is enabled and it starts counting up if the counter enable bit (bit 1) is high. This bit is cleared by reset.
Bit 1	When this bit is low, the counter is stopped. The counter value is not changed when the counter is stopped and started with this bit. When this bit is high, the counter is enabled. This bit is cleared by reset.
Bit 2	The overflow counter is cleared by writing a 1 to this bit.
Bits 4-7	These read only bits are the output of the overflow counter. The overflow counter is incremented each time the tick timer rolls over. These bits are cleared by reset.

AC Fail Interrupt Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE101C	IntStat	ACFail			Enable			
	R/C	R			R/W			

Bit 3	When this bit is high, the interrupt is enabled. The interrupt is disabled when this bit is low. This bit is cleared by reset.
Bit 6	When this bit is low, the VMEbus ACFAIL* signal is not active. When this bit is high, the VMEbus ACFAIL* signal is active.
Bit 7	When this bit is high, an AC Fail interrupt is being generated at level 7. This bit is edge sensitive and it is set on the leading edge of interrupt enable and AC Fail. This bit is cleared when a 1 is written to it or when the interrupt is disabled. When cleared, it remains cleared until the next leading edge of interrupt enable and AC Fail. This bit is cleared by reset.

Watchdog Timer Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE101D	WdL3	WdL2	WdL1	WdL0	WdTO	WdRst	WdClr	Enable
	R/W	R/W	R/W	R/W	R/C	R/W	C	R/W

Notes 1. Bit set and clear instructions should not be used on this control register. Because the WD time-out bit is cleared by writing a 1 to it and the status bit is a 1 to indicate a time-out, the read-modify-write sequence may clear the WD time-out.

2. The recommended use of this register is first to set all the control bits as desired but with the enable bit low. Next, write this register again with the same data, but this time with the enable bit high; this access will actually start the watchdog timer.

Bit 0	When this bit is low, the watchdog timer is disabled. When this bit is high, the watchdog timer is enabled, and increments each time tick timer 1 rolls over. This bit is cleared by reset.
Bit 1	The watchdog timer is cleared by writing a 1 to this bit.
Bit 2	When this bit is low, the watchdog timer does not activate the reset signal if a time-out occurs. When this bit is high, the watchdog timer activates the reset signal if a time-out occurs. This bit is cleared by reset. This bit should only be set if the MVME147 is system controller.
Bit 3	This bit is set if the watchdog timer times out. This bit is cleared by writing a 1 to it. This bit is cleared by reset.
Bits 4-7	These bits set the watchdog limit. When the watchdog timer value is equal to the watchdog limit, the WdTO bit (bit 3) is set. These bits are cleared by reset.

Printer Interrupt Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE101E	IntStat	FaltInt	ACKInt	ACKPol	Enable	IL2	IL1	IL0
	R	R/C	R/C	R/W	R/W	R/W	R/W	R/W

Note Bit set and clear instructions should not be used on this control register. Because the interrupt is cleared by writing a 1 to the status bit and the status bit is a 1 to indicate a pending interrupt, the read-modify-write sequence may clear a pending interrupt.

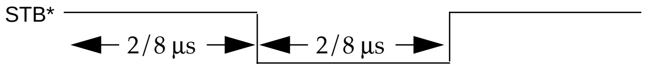
Bits 0-2	These bits program the interrupt level the printer generates. Level 0 does not generate an interrupt. These bits are cleared by reset.
Bit 3	When this bit is high, the interrupt is enabled. The interrupt is disabled when this bit is low. This bit is cleared by reset.
Bit 4	When this bit is low, the rising edge of ACK* generates an interrupt. When this bit is high, the falling edge of ACK* generates an interrupt. This bit is cleared by reset.
Bit 5	When interrupts are enabled, this bit is set by the rising or falling edge of ACK* as selected by bit 4. This bit is edge sensitive and is cleared by writing a 1 to it or when printer interrupts are disabled.
Bit 6	When interrupts are enabled, this bit is set by the falling edge of FAULT*. This bit is edge sensitive and is cleared by writing a 1 to it or when printer interrupts are disabled.
Bit 7	When this bit is high, a printer interrupt is being generated at the level programmed in bits 0-2. This bit is the OR of bits 5 and 6. This bit is cleared by reset.

Printer Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE101F					InPrim	Strobe	StbTim	Mode
					R/W	R/W	R/W	R/W

Bit 0 This bit selects the auto or manual mode for the printer strobe. When this bit is low, the printer strobe is generated by a write to the printer data register (auto mode). When this bit is high, the printer strobe is not generated by a write to the printer data register (manual mode). This bit is cleared by reset.

Bit 1 This bit controls the printer strobe timing in the auto mode. When this bit is low, the strobe time in the auto mode is 2 microseconds. When this bit is high, the strobe time in the auto mode is 8 microseconds. The strobe time is also the time delay from the write to the printer data register to the assertion of the printer strobe. This bit is cleared by reset.



Bit 2 This bit controls the printer strobe in the manual mode. In the manual mode, the software must control the timing. When this bit is low, the printer strobe is not activated. When this bit is high, the printer strobe is activated. This bit is cleared by reset.

Bit 3 This bit controls the Input Prime signal. When this bit is low, the Input Prime signal is not activated. When this bit is high, the Input Prime signal is activated. The software must control the timing of the printer Input Prime signal. This bit is cleared by reset.

DMA Interrupt Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE1020	IntStat				Enable	IL2	IL1	IL0
	R/C				R/W	R/W	R/W	R/W

Note Bit set and clear instructions should not be used on this control register. Because the interrupt is cleared by writing a 1 to the status bit and the status bit is a 1 to indicate a pending interrupt, the read-modify-write sequence may clear a pending interrupt.

Bits 0-2	These bits program the interrupt level the DMA controller generates. Level 0 does not generate an interrupt. These bits are cleared by reset.
Bit 3	When this bit is high, the interrupt is enabled. The interrupt is disabled when this bit is low. This bit is cleared by reset.
Bit 7	When this bit is high, a DMA interrupt is being generated at the level programmed in bits 0-2. This bit is edge sensitive and it is set on the leading edge of interrupt enable and DMA DONE (in DMA Control and Status Register). This bit is cleared when a 1 is written to it or when the interrupt is disabled. When cleared, it remains cleared until the next leading edge of interrupt enable and DMA DONE. This bit is cleared by reset.

DMA Control and Status Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE1021	DONE	8BitEr	TblSizEr	DMABEr	TWBEr	MS/SM*	TW	Enable
	R	R	R	R	R	R/W	R/W	R/W

Note All bits are cleared by reset.

Bit 0	When this bit is low, the DMA controller is disabled and status bits 3-7 are reset. When this bit is high, the DMA controller is enabled.
Bit 1	This bit controls the mode of the DMA controller. When this bit is low, the DMA controller uses the address and byte count in the address and byte count registers. When this bit is high, the DMA controller uses address and byte counts in a table pointed to by the table address register.
Bit 2	This bit controls the direction the data is transferred. When this bit is low, the DMA controller transfers data from the SCSI bus. When this bit is high, the DMA controller transfers data to the SCSI bus.
Bit 3	This bit is set if a bus error occurred while the DMA controller was accessing the address table. This bit is reset when the DMA controller is disabled.
Bit 4	This bit is set if a bus error occurred while the DMA controller was transferring data. This bit is reset when the DMA controller is disabled.
Bit 5	This bit is set if the DMA controller accesses a table entry that is not located in 32-bit memory. This bit is reset when the DMA controller is disabled.
Bit 6	This bit (8-bit error) is set if the DMA controller receives a handshake indicating the port was 8 bits. This bit is reset when the DMA controller is disabled.
Bit 7	This bit is set when the DMA controller has stopped because all the data has been transferred or an error has occurred. This bit is reset when the DMA controller is disabled.

Bus Error Interrupt Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE1022	IntStat				Enable			
	R/C				R/W			

Bit 3	When this bit is high, the interrupt is enabled. The interrupt is disabled when this bit is low. This bit is cleared by reset.
Bit 7	When this bit is high, a bus error interrupt is being generated at Level 7. This bit is set when the processor receives a bus error and the interrupt is enabled. This bit is cleared when a 1 is written to it or when the interrupt is disabled. When cleared, it remains cleared until the next bus error. This bit is cleared by reset.

DMA Status Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE1023	Inc 4	Inc 3	Inc 2	Inc 1	UU	UM	LM	LL
	R	R	R	R	R	R	R	R

Bits 0-3	The PCC has a 32-bit register which is used to hold data that is transferred between the SCSI bus and the local bus. Bits 0-3 indicate the status of each byte of the holding register (byte position in longword: UU = upper upper; UM = upper middle; LM = lower middle; LL = lower lower). When a bit is low, the corresponding byte is empty. When a bit is high, the corresponding byte is full. These bits are cleared when the DMA controller is disabled. These bits are cleared by reset.
Bits 4-7	The DMA address and byte counters may be incremented by 1, 2, 3, or 4. When the DMA counters are incremented, the increment value is saved in these bits. Only one of the 4 bits is set. These bits are cleared when the DMA controller is disabled. These bits are cleared by reset.

Abort Interrupt Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE1024	IntStat	Abort			Enable			
	R/C	R			R/W			

Note Bit set and clear instructions should not be used on this control register. Because the interrupt is cleared by writing a 1 to the status bit and the status bit is a 1 to indicate a pending interrupt, the read-modify-write sequence may clear a pending interrupt.

Bit 3	When this bit is high, the interrupt is enabled. The interrupt is disabled when this bit is low. This bit is cleared by reset.
Bit 6	This bit indicates the current state of the ABORT switch. When this bit is low, the ABORT switch is not pressed. When this bit is high, the ABORT switch is pressed.
Bit 7	When this bit is high, an abort interrupt is being generated at Level 7. This bit is edge sensitive and it is set on the leading edge of interrupt enable and abort. This bit is cleared when a 1 is written to it or when the interrupt is disabled. When cleared, it remains cleared until the next leading edge of interrupt enable and abort. This bit is cleared by reset.

Table Address Function Code Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE1025						TblFC2	TblFC1	TblFC0
						R/W	R/W	R/W

Bits 0-2 This function code is placed on the local bus when the DMA address table is accessed. Note that a value of 1, 2, 5, or 6 must be placed in Tbl FC2-FC0 for proper operation of the MVME147 during table walking. These bits are cleared by reset.

Note DMA is enabled by the DMA Control and Status Register.

Serial Port Interrupt Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE1026	IntStat			Int/Ext*	Enable	IL2	IL1	IL0
	R			R/W	R/W	R/W	R/W	R/W

Note All bits are cleared by reset.

Bits 0-2	These bits program the interrupt level that the serial ports generate. Level 0 does not generate an interrupt.
Bit 3	When this bit is high, the interrupt is enabled. The interrupt is disabled when this bit is low.
Bit 4	This bit controls the vector source. When this bit is low, the interrupt status/id vector comes from the serial chip. When this bit is high, the interrupt status/id vector comes from the PCC.
Bit 7	When this bit is high, a serial port interrupt is being generated at the level programmed in bits 0-2. This bit is level sensitive and it is active when interrupt enable and serial port interrupt are active.

General Purpose Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE1027	RsDis2	RsDis1	RsDis0	MIntEn	LbToEn	WWPar	ParEn1	ParEn0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bits 0-1	These bits control local RAM parity checking. These bits should not be enabled on the MVME147-010. These bits are cleared by reset.

Note The DRAM parity on the MVME147 is in an undefined state after power-up. Reads to uninitialized memory with parity checking enabled causes bus errors. All DRAM locations should be written to ensure correct parity before checking is enabled.

0	Local RAM parity checking is disabled.
1	Local RAM parity checking is enabled and BERR is asserted during the current DRAM access cycle (adds 1 wait cycle).
2	Local RAM parity checking is disabled.
3	Local DRAM parity checking is enabled. BERR is asserted on the current cycle (adds 1 wait cycle) for LANCE, VME, and PCC accesses to DRAM. BERR is asserted on the next DRAM access cycle for MC68030 accesses to DRAM (adds 0 wait cycles). Note that not only is BERR asserted during the next MC68030 DRAM access cycle but it is asserted during all subsequent MC68030 DRAM access cycles. This helps stop the MC68030 from proceeding when DRAM is bad.

Bit 2	This bit is used to test the parity generating and checking logic. When this bit is low, correct parity is written to the DRAM; when high, incorrect parity is written to the DRAM. This bit is cleared by reset.
Bit 3	When set, this bit is used to enable the local bus timer that is part of the PCC. Because the VMEchip also contains a local bus timer, this bit should be cleared, turning off the PCC local bus timer. This bit is cleared by reset.
Bit 4	This bit is the master interrupt enable. When this bit is low, all interrupts on the MVME147 are disabled; when high, all interrupts are enabled. This bit is cleared by reset.
Bits 5-7	When the pattern %101 is written to these bits, the front panel RESET switch is disabled. The RESET switch is enabled for any other pattern. These bits are cleared by reset.

LAN Interrupt Control Register

Note The LAN interrupt is not used on the MVME147-010 and should not be enabled.

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE1028	IntStat				Enable	IL2	IL1	IL0
	R				R/W	R/W	R/W	R/W

Note All bits are cleared by reset.

Bits 0-2	These bits program the interrupt level the LAN chip generates. Level 0 does not generate an interrupt.
Bit 3	When this bit is high, the interrupt is enabled. The interrupt is disabled when this bit is low.
Bit 7	When this bit is high, a LAN port interrupt is being generated at the level programmed in bits 0-2. This bit is level sensitive and it is active when interrupt enable and LAN interrupt are active.

General Purpose Status Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE1029							PuReset	ParErr
							R/C	R/C

Bit 0	This bit is set when a parity error occurs while the local processor is accessing RAM. This bit is cleared by writing a 1 to it. This bit is cleared by reset.
Bit 1	This bit is set when a power-up reset occurs. It is cleared by writing a 1 to it. When the MVME147BUG is installed, its initialization code clears this bit.

SCSI Port Interrupt Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE102A	IntStat	RstInt	SCSIRst	RstSCSI	Enable	IL2	IL1	IL0
	R	R/C	R	R/W	R/W	R/W	R/W	R/W

Note Bit set and clear instructions should not be used on this control register. Because the interrupt is cleared by writing a 1 to status bit and the status bit is a 1 to indicate a pending interrupt, the read-modify-write sequence may clear a pending interrupt.

Bits 0-2	These bits program the interrupt level the SCSI port generates. Level 0 does not generate an interrupt. These bits are cleared by reset.
Bit 3	When this bit is high, the interrupt is enabled. The interrupt is disabled when this bit is low. This bit is cleared by reset.
Bit 4	This bit is used to control the reset signal on the SCSI bus. When this bit is low, the SCSI reset signal is not driven by MVME147. When this bit is high, the SCSI reset is driven by MVME147. This bit is cleared by reset.
Bit 5	This bit indicates the state of the SCSI reset signal. When this bit is low, the SCSI reset signal is not active. When this bit is high, the SCSI reset signal is active.
Bit 6	When this bit is high, a SCSI reset interrupt is being generated at the level programmed in bits 0-2. This bit is edge sensitive and it is set on the leading edge of interrupt enable and SCSI reset. This bit is cleared when a 1 is written to it or when the interrupt is disabled. When cleared, it remains cleared until the next leading edge of interrupt enable and SCSI reset. This bit is cleared by reset.
Bit 7	When this bit is high, a SCSI port interrupt is being generated at the level programmed in bits 0-2. This bit is the OR of bit 6 and the SCSI chip interrupt. This bit is cleared by reset.

Slave Base Address Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE102B	LANA25	LANA24	WAITRMC	RBA4	RBA3	RBA2	RBA1	RBA0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Note All bits are cleared by reset.

Bits 0-4 These bits set the slave RAM base address, or the address of onboard RAM as viewed from the VMEbus.

Table 4-2. DRAM Address as Viewed from the VMEbus

RBA4	RBA3	RBA2	RBA1	RBA0	Beginning Address	Ending Address	Notes
0	0	0	0	0	$00000000	(1 x DRAMsize)-1	
0	0	0	0	1	1 x DRAMsize	(2 x DRAMsize)-1	1, 2
0	0	0	1	0	2 x DRAMsize	(3 x DRAMsize)-1	1, 2
0	0	0	1	1	3 x DRAMsize	(4 x DRAMsize)-1	1, 2
0	0	1	0	0	4 x DRAMsize	(5 x DRAMsize)-1	1, 2
0	0	1	0	1	5 x DRAMsize	(6 x DRAMsize)-1	1, 2
0	0	1	1	0	6 x DRAMsize	(7 x DRAMsize)-1	1, 2
0	0	1	1	1	7 x DRAMsize	(8 x DRAMsize)-1	1, 2
0	1	0	0	0	8 x DRAMsize	(9 x DRAMsize)-1	1, 2
0	1	0	0	1	9 x DRAMsize	(10 x DRAMsize)-1	1, 2
0	1	0	1	0	10 x DRAMsize	(11 x DRAMsize)-1	1, 2
0	1	0	1	1	11 x DRAMsize	(12 x DRAMsize)-1	1, 2
0	1	1	0	0	12 x DRAMsize	(13 x DRAMsize)-1	1, 2
0	1	1	0	1	13 x DRAMsize	(14 x DRAMsize)-1	1, 2
0	1	1	1	0	14 x DRAMsize	(15 x DRAMsize)-1	1, 2
0	1	1	1	1	15 x DRAMsize	(16 x DRAMsize)-1	1, 2

Table 4-2. DRAM Address as Viewed from the VMEbus (Continued)

RBA4	RBA3	RBA2	RBA1	RBA0	Beginning Address	Ending Address	Notes
1	0	0	0	0	16 x DRAMsize	(17 x DRAMsize)-1	1, 2
1	0	0	0	1	17 x DRAMsize	(18 x DRAMsize)-1	1, 2
1	0	0	1	0	18 x DRAMsize	(19 x DRAMsize)-1	1, 2
1	0	0	1	1	19 x DRAMsize	(20 x DRAMsize)-1	1, 2
1	0	1	0	0	20 x DRAMsize	(21 x DRAMsize)-1	1, 2
1	0	1	0	1	21 x DRAMsize	(22 x DRAMsize)-1	1, 2
1	0	1	1	0	22 x DRAMsize	(23 x DRAMsize)-1	1, 2
1	0	1	1	1	23 x DRAMsize	(24 x DRAMsize)-1	1, 2
1	1	0	0	0	24 x DRAMsize	(25 x DRAMsize)-1	1, 2
1	1	0	0	1	25 x DRAMsize	(26 x DRAMsize)-1	1, 2
1	1	0	1	0	26 x DRAMsize	(27 x DRAMsize)-1	1, 2
1	1	0	1	1	27 x DRAMsize	(28 x DRAMsize)-1	1, 2
1	1	1	0	0	$00000000	(1 x DRAMsize)-1	1, 3, 4
1	1	1	0	1	1 x DRAMsize	(2 x DRAMsize)-1	1, 3, 4

Notes

1. DRAMsize = the size of the DRAM. For example, if the 4MB version is used, then DRAMsize = $400000, and (3 x DRAMsize)-1 = $BFFFFF.

2. When beginning address is less then 16MB, the DRAM responds to standard or extended address modifiers. When beginning address is 16MB or greater, the DRAM responds to extended address modifiers only. Note that bits 4 and 5 in the VMEchip Slave Address Modifier Register further control response to standard and extended address modifiers.

3. This combination pertains only to DRAMsize of 16MB or 32MB.

4. The values shown in the table refer to extended addresses only. In the standard address range the DRAM responds to $000000 through $7FFFFF.

Bit 5 WAITRMC controls the MVME147 implementation of multiple address RMC (MARMC) cycles. When WAITRMC is set, the MVME147 always waits for VMEbus mastership before executing an MARMC cycle. WAITRMC should be set if it is desired to guarantee indivisibility of MARMC cycles (only guaranteed if the other master implements MARMC cycles the same way as the MVME147).

When WAITRMC is cleared, the MVME147 only waits for VMEbus mastership if the MARMC cycle starts out by going to the VMEbus.

Note Regardless of the state of the WAITRMC bit, if the MVME147 obtains VMEbus mastership during an MARMC, it maintains it until all of the cycles of the MARMC are completed.

WAITRMC operation is effective only if MASWP bit in the VMEchip LCSR (bit 5 $FFFE2005) is cleared (MASWP posting is disabled.)

Bits 6-7 These bits determine the section of local DRAM that is accessible to the LANCE during DMA.

Table 4-3. DRAM Accessed by the LANCE

LANA25	LANA24	Section of DRAM Accessible to LANCE
0	0	$00000000-00FFFFFF
0	1	$01000000-01FFFFFF
1	0	$02000000-02FFFFFF
1	1	$03000000-03FFFFFF

Software Interrupt 1 Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE102C	IntStat				Enable	IL2	IL1	IL0
	R				R/W	R/W	R/W	R/W

Note All bits are cleared by reset.

Bits 0-2	These bits program the interrupt level that is generated. Level 0 does not generate an interrupt.
Bit 3	When this bit is high, the interrupt is enabled. The interrupt is disabled when this bit is low.
Bit 7	This bit is low when the interrupt is disabled and it is high when the interrupt is enabled.

Interrupt Vector Base Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE102D	IVB7	IVB6	IVB5	IVB4				
	R/W	R/W	R/W	R/W				

Bits 4-7 These bits are used to form the base interrupt status/ID vector for interrupts whose vectors originate from the PCC. The lower four bits of the vector are determined by the interrupting device. These bits are cleared by reset.

Bits 3-0

	3	2	1	0
AC Fail	0	0	0	0
BERR	0	0	0	1
Abort	0	0	1	0
Serial port (when enabled by PCC)	0	0	1	1
LANCE	0	1	0	0
SCSI port	0	1	0	1
SCSI DMA	0	1	1	0
Printer port	0	1	1	1
Tick timer 1	1	0	0	0
Tick timer 2	1	0	0	1
Software interrupt 1	1	0	1	0
Software interrupt 2	1	0	1	1

Note The serial port interrupt status/ID vector source is determined by bit 4 of the Serial Port Interrupt Control Register ($FFFE1026).

Software Interrupt 2 Control Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE102E	IntStat				Enable	IL2	IL1	IL0
	R				R/W	R/W	R/W	R/W

Note All bits are cleared by reset.

Bits 0-2	These bits program the interrupt level that is generated. Level 0 does not generate an interrupt.
Bit 3	When this bit is high, the interrupt is enabled. The interrupt is disabled when this bit is low.
Bit 7	This bit is low when the interrupt is disabled and it is high when the interrupt is enabled.

Revision Level Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE102F	RevL7	RevL6	RevL5	RevL4	RevL3	RevL2	RevL1	RevL0
	R	R	R	R	R	R	R	R

Bits 0-7	These bits represent the revision level of the PCC. Initial parts are released as level 0. If functional changes are required in future parts, the revision level is incremented. This allows the software to configure itself should functional changes be required in the PCC.

Printer Data Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2800	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
	W	W	W	W	W	W	W	W

Bits 0-7	These bits form the printer data lines. They are write only. Reading this address accesses the printer status register. These bits are not affected by reset.

Printer Status Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2800	ACK	FAULT	SELECT	PE	BSY		LOW	STAT12
	R	R	R	R	R		R	R

Note All of these bits are read only. Writing this address accesses the printer data register. These bits are not affected by reset.

Bit 0	STAT12 indicates the status of the fused +12V power for Ethernet transceiver power and for serial port pull up power.
Bit 1	LOW is always 0.
Bit 3	BSY is 1 when the printer is busy and 0 when it is not.
Bit 4	PE is 1 when the printer is in the paper empty state and 0 when it is not.
Bit 5	SELECT is 1 when the printer is selected and 0 when it is not.
Bit 6	FAULT is 1 when the printer is in the fault state and 0 when it is not.
Bit 7	ACK is 1 when printer acknowledge is true and 0 when it is not.

Programming the VMEchip

The VMEchip has two groups of registers: the Local Control and Status Registers (LCSR) and the Global Control and Status Registers (GCSR).

Programming the LCSR

There are 14 LCSR registers as shown in Table 4-4.

Table 4-4. VMEchip Local Control and Status Registers

Address	Register								Name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
FFFE2001					ROBIN	BRDFAIL	SRESET	SCON	Sys. Controller Cnfg.
FFFE2003	DWB	DHB	RONR	RWD	RNEVER		RQLEV1	RQLEV0	VMEbus Req'r Cnfg.
FFFE2005	DDTACK	020	MASWP	CFILL	MASUAT	MASA16	MASA24	MASD16	Master Configuration
FFFE2007	SLVEN		SLVWP					SLVD16	Slave Configuration
FFFE2009		ARBTO	VBTO1	VBTO0	ACTO1	ACTO0	LBTO1	LBTO0	Timer Configuration
FFFE200B	SUPER	USER	EXTED	STND	SHORT	BLOCK	PRGRM	DATA	Slave Address Mod.
FFFE200D	AMSEL		AM5	AM4	AM3	AM2	AM1	AM0	Master Address Mod.
FFFE200F	IEN7	IEN6	IEN5	IEN4	IEN3	IEN2	IEN1		Int. Handler Mask
FFFE2011	WPERREN	SFIEN	SIGHEN	LM1EN	IACKEN	LM0EN	SIGLEN		Utility Int. Mask
FFFE2013	UVB7	UVB6	UVB5	UVB4	UVB3	UID2	UID1	UID0	Utility Int. Vector
FFFE2015						IL2	IL1	IL0	Interrupt Request
FFFE2017	D07	D06	D05	D04	D03	D02	D01	D00	VMEbus Status/ID
FFFE2019					RMCERR	VBERR	ACTO	LBTO	Bus Error Status
FFFE201B					GCSRA7	GCSRA6	GCSRA5	GCSRA4	GCSR Base Ad. Cnfg.

Note The bottom line in the table for each of the following LCSR register definitions defines the operations possible on the register bits, as follows:

R	This bit is a read-only status bit.
R/W	This bit is readable and writable.
C	Writing a 1 to this bit clears it. This bit reads 0.
R/C	This bit is readable. Writing a 1 to this bit clears it.

System Controller Configuration Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2001					ROBIN	BRDFAIL	SRESET	SCON
					R/W	R/W	W	R

Note These bits are not affected by reset.

Bit 0	The SCON status bit is a reflection of the configuration of header J3. When J3 pins 1 and 2 are connected, enabling the MVME147 to act as the VMEbus system controller, then SCON = 1. When J3 pins 1 and 2 are not connected, the MVME147 is not the VMEbus system controller and SCON = 0.
Bit 1	This bit allows the software to initiate a global reset sequence. Setting the SRESET bit activates the SYSRESET* signal on the VMEbus which in turn resets the MVME147. This bit clears automatically after the reset is complete. This bit is cleared by any reset.
Bit 2	Setting BRDFAIL to 1 causes the VMEchip to attempt to activate the SYSFAIL* signal on the VMEbus. The GCSR bit Inhibit SYSFAIL (ISF), in global register 1, enables the MVME147 to cause SYSFAIL* to be activated as a result of the state of BRDFAIL. In addition, when the bit is set, the FAIL LED is lit. (A watchdog time-out from the PCC also lights the FAIL LED.) This bit is set by any reset.
Bit 3	The ROBIN bit configures the VMEbus arbitration mode. ROBIN = 1 forces the round-robin mode. ROBIN = 0 forces the priority mode. Both modes can be used by the MVME147. This bit is cleared by SYSRESET.

VMEbus Requester Configuration Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2003	DWB	DHB	RONR	RWD	RNEVER		RQLEV1	RQLEV0
	R/W	R	R/W	R/W	R/W		R/W	R/W

Bits 0-1 These control bits configure the VMEbus requester level as shown in the table below:

RQLEV1	**RQLEV0**	**Level**
0	0	0
0	1	1
1	0	2
1	1	3

These bits are set to 1, 1 by any reset.

Note that writes to REQLEV1,0 do not change the actual requester level until the MVME147 goes through the action of having VMEbus mastership and releasing it. This means that there are times when the value written into REQLEV1,0 do not match the current requester level (the request level is lagging). During such times, reads to REQLEV1,0 reflect the actual requester level, not the value written into REQLEV1,0.

Bit 3 Setting this bit to 1 prevents the requester from releasing the VMEbus. However, unlike the DWB control bit, setting the RNEVER bit does not cause the requester to request the VMEbus. Clearing the RNEVER bit allows the requester to relinquish the VMEbus in accordance with the other control bits of the requester configuration register. This bit is cleared by any reset.

Bit 4	The RWD bit allows software to configure the requester release mode. When the bit is set, if RNEVER and DWB are both cleared to 0, the requester releases the VMEbus after the MC68030 completes a VMEbus cycle. When the bit is cleared, if RNEVER and DWB are both cleared to 0, the requester operates in the Release-On-Request (ROR) mode. After acquiring control of the VMEbus, it maintains control until it detects another request pending on the VMEbus. This bit is cleared by any reset.
Bit 5	The RONR bit controls the manner in which the VMEchip requests the VMEbus. When the bit is set; anytime the MVME147 has bus mastership, then gives it up, the VMEchip does not request the VMEbus again until it detects the bus request signal BR*, on its level, negated for at least 150 ns. When the VMEchip detects BR* negated, it refrains from driving it again for at least 200 ns. This bit is cleared by any reset.
Bit 6	The DHB status bit is 1 when the MVME147 is VMEbus master and 0 when it is not.
Bit 7	Setting the DWB control bit to 1 causes the VMEchip to request the VMEbus (if not already bus master). When VMEbus mastership has been obtained, it is not relinquished until after the DWB and RNEVER bits are both cleared. This bit is cleared by any reset.

Master Configuration Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2005	DDTACK	020	MASWP	CFILL	MASUAT	MASA16	MASA24	MASD16
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit 0	Setting the MASD16 bit forces the MVME147 to perform only D8 and D16 data transfers on the VMEbus. Clearing the MASD16 bit allows D8, D16, and D32 transfer capability on the VMEbus when the MC68030 accesses in the range below $F0000000. (Accesses to VMEbus locations above $F0000000 are always restricted to D8/D16 regardless of the MASD16 bit.) This bit is cleared by SYSRESET.
Bit 1	If either the MASA24 bit is set, or the MC68030 accesses the VMEbus in the range below $1000000, the master drives one of the standard (24-bit) address modifier codes during VMEbus cycles (unless the master is configured to use the master address modifier register as described in the *Master Address Modifier Register* section in this chapter). The specific standard AM code is determined from the levels that the MC68030 drives on the three function code lines during the cycle, as shown in the table below. This bit is cleared by SYSRESET.
Bit 2	If either the MASA16 bit is set, or the MC68030 accesses the VMEbus in the range above $FFFF0000, a short (16-bit) AM code is used regardless of the state of the MASA24 bit (unless the master is configured to use the master address modifier register as described in the *Master Address Modifier Register* section in this chapter). The specific short AM code is determined from the levels that the MC68030 drives on the three function code lines during the cycle, as shown in the following table. This bit is cleared by SYSRESET.

Table 4-5. Determining the Master AM Code

VMEbus Address Modifier														
MASA16	ADR16	MASA24	ADR24	FC2	FC1	FC0		AM5	AM4	AM3	AM2	AM1	AM0	Code
0	F	0	F	0	0	1		0	0	1	0	0	1	$09
0	F	0	F	0	1	0		0	0	1	0	1	0	$0A
0	F	0	F	1	0	1		0	0	1	1	0	1	$0D
0	F	0	F	1	1	0		0	0	1	1	1	0	$0E
1	X	X	X	0	0	1		1	0	1	0	0	1	$29
1	X	X	X	0	1	0		1	0	1	0	1	0	$2A
X	T	X	X	0	0	1		1	0	1	0	0	1	$29
X	T	X	X	0	1	0		1	0	1	0	1	0	$2A
1	X	X	X	1	0	1		1	0	1	1	0	1	$2D
1	X	X	X	1	1	0		1	0	1	1	1	0	$2E
X	T	X	X	1	0	1		1	0	1	1	0	1	$2D
X	T	X	X	1	1	0		1	0	1	1	1	0	$2E
0	F	1	X	0	0	1		1	1	1	0	0	1	$39
0	F	1	X	0	1	0		1	1	1	0	1	0	$3A
0	F	1	X	1	0	1		1	1	1	1	0	1	$3D
0	F	1	X	1	1	0		1	1	1	1	1	0	$3E
0	F	X	T	0	0	1		1	1	1	0	0	1	$39
0	F	X	T	0	1	0		1	1	1	0	1	0	$3A
0	F	X	T	1	0	1		1	1	1	1	0	1	$3D
0	F	X	T	1	1	0		1	1	1	1	1	0	$3E
T = True, F = False, X = Don't Care														

Notes

AM2, 1, 0 track FC2, 1, 0.
ADR16 = T represents MC68030 accesses to the VMEbus above $FFFF0000.
ADR16 = F represents MC68030 accesses to the VMEbus below $FFFF0000.
ADR24 = T represents MC68030 accesses to the VMEbus below $01000000.
ADR24 = F represents MC68030 accesses to the VMEbus above $01000000.

Bit 3	The MASUAT bit allows software to configure the master to provide the UAT data transfer capability. Setting the MASUAT bit to 1 configures the master to execute unaligned VMEbus cycles when necessary. If the bit is cleared, the MC68030 is acknowledged so as to break the unaligned transfer into multiple aligned cycles. This bit is cleared by SYSRESET.
Note	While making it optional for the master to provide the UAT data transfer capability, the VMEbus specification requires that all D32 slaves support it.
Bit 4	This bit is cleared by SYSRESET. It should remain cleared.
Bit 5	Setting the MASWP bit speeds up MC68030 writes to the VMEbus. However, it should be used with caution. When MASWP (Master Write Posting) is set, MC68030 write cycles to the VMEbus are acknowledged by the VMEchip, before they have actually finished on the VMEbus. The VMEchip finishes the write cycles on its own, allowing the MC68030 to continue with new cycles. If the SLVEN bit is cleared (slave disabled), the VMEchip acknowledges VME writes even before it has obtained VMEbus mastership. If the SLVEN bit is set, then it waits until it has obtained VMEbus mastership. This bit is cleared by SYSRESET.
Note	The MC68030 is not notified via BERR* if an error occurs while the VMEchip is finishing a write posted cycle. The VMEchip can be programmed to interrupt the MC68030 if such an event occurs (WPERREN bit in the Utility Interrupt Mask Register). Keep in mind that interrupt notification could be well after the occurrence of the error.
Bit 6	020 - This bit should always be cleared.
Bit 7	DDTACK - This bit should always be cleared for 25 MHz boards and set for 32 MHz boards.

Slave Configuration Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2007	SLVEN		SLVWP					SLVD16
	R/W		R/W					R/W

Note The bits in the slave configuration must be changed only when the VMEchip has control of the VMEbus. The recommended procedure for changing the slave configuration is:

a. Set the DWB bit in the requester configuration register to 1.

b. Read the DHB status bit until it is 1.

c. Change the slave configuration register.

d. Clear the DWB bit to 0.

Bit 0	SLVD16 should always be cleared. Setting SLVD16 to 1 configures the VMEchip slave to provide only D08 (EO) and D16 data transfer capabilities. It is typically set when the local bus is only 16 bits wide. Clearing the SLVD16 bit to 0 configures the VMEchip slave to provide the D08 (EO), D16, and D32/UAT data transfer capabilities. This bit is cleared by SYSRESET.
Bit 5	Setting the SLVWP bit speeds up VMEbus writes to the onboard DRAM. When SLVWP (slave write posting) is set, VMEbus write cycles to the onboard DRAM are acknowledged by the VMEchip before the data has been written into the DRAM. This allows the VMEbus master to end its cycle quickly, placing the burden on the VMEchip to complete the write to onboard DRAM on its own. This bit is cleared by SYSRESET.
Bit 7	Setting SLVEN to 1 enables other VMEbus masters to access the MVME147 onboard DRAM. This bit is cleared by SYSRESET.

Timer Configuration Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2009		ARBT0	VBTO1	VBTO0	ACTO1	ACTO0	LBTO1	LBTO0
		R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bits 0-1 These two bits configure the local time-out period. They are set to 1 by any reset.

LBTO1	**LBTO0**	**Time-Out Period**
0	0	102 microseconds
0	1	205 microseconds
1	0	410 microseconds
1	1	Timer disabled

The local bus timer activates bus error to the MC68030 when it tries to access nonexistent locations in the local memory map.

Bits 2-3 These two bits configure the VMEbus access time-out period. They are set to 1 by any reset.

ACTO1	**ACTO0**	**Time-Out Period**
0	0	102 microseconds
0	1	1.6 millisecond
1	0	51 milliseconds
1	1	Timer disabled

The VMEbus access timer activates bus error to the MC68030 (except on write posted time-outs) when the VMEchip is unsuccessful in obtaining the VMEbus within the time-out period.

Bits 4-5 These two bits configure the VMEbus global time-out period. VBTO1 is set to 1 and VBTO0 is cleared to 0 by SYSRESET.

VBTO1	**VBTO0**	**Time-Out Period**
0	0	102 microseconds
0	1	205 microseconds
1	0	410 microseconds
1	1	Timer disabled

The VMEbus global timer activates BERR* on the VMEbus.

Bit 6 Setting ARBTO to 1 enables the VMEbus arbitration timer. The VMEbus arbitration timer activates BBSY* if it is not activated within 410 µs after the MVME147 arbiter issues a bus grant. The timer deactivates BBSY* as specified in the VMEbus specification. This causes the arbiter to arbitrate any pending requests for the bus. This bit is set to 1 by SYSRESET.

Slave Address Modifier Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE200B	SUPER	USER	EXTED	STND	SHORT	BLOCK	PRGRM	DATA
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Note This register allows software to configure which address modifier codes the VMEbus masters must use to access the onboard DRAM. The 8 bits of the register are organized into three groups. At least one of the bits in each group must be set, otherwise the address modifier used by the master is ignored.

Bits 0-2 These three bits form the first group which configures the slave AM code. Setting any of the bits to 1 enables the slave to respond to cycles as described in the example below. Note BLOCK should never be set. These bits are cleared by SYSRESET.

Bits 3-5 These three bits form the second group. Setting any of the bits to 1 enables the slave to respond to cycles as described in the example below. These bits are cleared by SYSRESET.

Bits 6-7 These two bits form the third group. Setting any of the bits to 1 enables the slave to respond to cycles as described in the example below. These bits are cleared by SYSRESET.

Example: If the SUPER, STND, and DATA bits are set, then the only AM code accepted is $3D, standard supervisor data access. When more than one bit is set in a group, the accepted AM codes include all permutations of the bits that are set. For example, if the SUPER, USER, EXTED, PRGRM, and DATA bits are set, the accepted AM codes are $09, $0A, $0D, and $0E. These are extended user data access, extended user program access, extended supervisor data access, and extended supervisor program access. The normal recommended configuration of the bits is all set except for BLOCK ($FB).

Note Although all bits in the slave address modifier register may be changed dynamically, they must be changed only when the VMEchip has control of the VMEbus.

The recommended procedure for changing the slave address modifier is:

a. Set the DWB bit in the requester configuration register to 1.

b. Read the DHB status bit until it is 1.

c. Change the slave address modifier register.

d. Clear the DWB bit to 0.

Master Address Modifier Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE200D	AMSEL		AM5	AM4	AM3	AM2	AM1	AM0
	R/W		R/W	R/W	R/W	R/W	R/W	R/W

Note The register allows software to program the address modifier code that is driven by the MVME147 during a VMEbus cycle.

Bits 0-5 These five bits, in conjunction with AMSEL, allow software to select dynamically the address space that the master accesses during VMEbus cycles. Setting any of these five bits to 1 causes the master to drive the corresponding address modifier line to high (if the AMSEL bit is set to 1).

Clearing any of the bits to 0 causes the master to drive the corresponding line to low (if the AMSEL bit is set to 1).

These bits are cleared by SYSRESET.

Bit 7 Software uses the AMSEL control bit to define what is the source of the AM code driven by the master during a VMEbus cycle.

Setting the bit to 1 causes the master to drive the contents of the lower six bits onto the address modifier lines. No attempt is made to check the value stored in this register for reserved or illegal address modifiers.

Clearing the AMSEL bit causes the master to determine the AM code dynamically.

AMSEL should normally be cleared to 0. This bit is cleared by SYSRESET.

Interrupt Handler Mask Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE200F	IEN7	IEN6	IEN5	IEN4	IEN3	IEN2	IEN1	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Notes This register is used to enable the MC68030 to respond to specific VMEbus interrupt requests. Note that the master interrupt enable bit in the PCC must also be set for VMEbus IRQs to get through to the MC68030.

Setting any of bits 1 through 7 unmasks an interrupt request from the VMEbus IRQ signal at the corresponding level. Keep in mind that only one VMEbus master is allowed to handle each level of VMEbus IRQ. The software should set these bits accordingly. These bits are cleared by any reset.

Utility Interrupt Mask Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2011	WPERREN	SFIEN	SIGHEN	LM1EN	IACKEN	LM0EN	SIGLEN	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Note This register is used to enable the VMEchip interrupt handler to respond to specific utility interrupt requests. When the interrupt handler detects an interrupt request from one of the enabled functions, it responds by requesting the MC68030 to initiate an interrupt acknowledge cycle if the master interrupt enable bit is set in the PCC. All the bits in this register are cleared by any reset.

Bit 1 As described in the *Programming the GCSR* section in this chapter, the GCSR provides two global attention interrupt bits: SIGLP and SIGHP, which allow other VMEbus masters to interrupt the MC68030 on a low priority (Level 1) and on a high priority (Level 5). Setting the SIGLEN control bit to 1 unmasks the SIGLP interrupt.

Bit 2 As described in the *Programming the GCSR* section in this chapter, the GCSR provides four location monitors. Two of them, location monitor 0 and 1, cause a local interrupt when the VMEbus address they are configured to monitor is accessed. The LM0EN control bit allows software to mask the interrupt requested when an access is detected to the address monitored by location monitor 0. The level of local interrupt is shown in Table 4-6. Setting the LM0EN bit to 1 unmasks the interrupt.

Bit 3 The VMEchip allows software to program the interrupt handler to generate a local interrupt after it concludes a VMEbus IACK cycle. The level of the local interrupt is shown in Table 4-6. Setting the IACKEN control bit to 1 enables the IACK interrupt.

This function is intended to be coupled with the use of the VMEchip global interrupt function. If this bit is set, a local interrupt (to the MC68030) is generated when a VMEbus IACK cycle acknowledges the interrupt (refer to the *Interrupt Request Register* section in this chapter).

Bit 4	As described in the *Programming the GCSR* section in this chapter, the GCSR provides four location monitors. Two of them, location monitor 0 and 1, cause a local interrupt when the VMEbus address they are configured to monitor is accessed. The LM1EN control bit allows software to mask the interrupt requested when an access is detected to the address monitored by location monitor 1. The level of local interrupt is shown in Table 4-6. Setting the LM1EN bit to 1 unmasks the interrupt.
Bit 5	As described in the *Programming the GCSR* section in this chapter, the GCSR provides a global high priority attention interrupt bit SIGHP which allows other VMEbus masters to interrupt the MC68030. The level of the local interrupt is shown in Table 4-6. Setting the SIGHEN control bit to 1 unmasks the SIGHP interrupt.
Bit 6	Setting SFIEN to 1 enables a low level on the VMEbus SYSFAIL* line to cause an interrupt to the MC68030. The level of the SYSFAIL* interrupt is shown in Table 4-6.
Bit 7	The VMEchip allows software to configure the VMEbus master to operate in a write posted mode (i.e., acknowledge the MC68030 VMEbus bound write cycle before it has actually been executed on the VMEbus). If the VMEchip encounters a VMEbus bus error as it attempts to complete the write posted cycle, the VMEchip notifies the MC68030 via Level 7 interrupt if the WPERREN bit is set.

Table 4-6. Utility Interrupts and Their Assigned Level

Utility Interrupt	Assigned Priority
SIGLP	Level 1
LM0	Level 2
IACK	Level 3
LM1	Level 4
SIGHP	Level 5
SYSFAIL	Level 6
WPBERR	Level 7

Utility Interrupt Vector Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2013	UVB7	UVB6	UVB5	UVB4	UVB3	UID2	UID1	UID0
	R/W	R/W	R/W	R/W	R/W	R	R	R

Notes The utility interrupt vector register provides the local CPU with a unique vector for each of the utility interrupts. Close examination reveals that the assigned level of each of the utility interrupts, as defined in the *Utility Interrupt Mask Register* section in this chapter, is the same as its assigned ID. This is implemented by reflecting the state of the address lines A01-A03, that the local CPU drives when it acknowledges an interrupt, onto bits 0-2 of the utility vector register. When accessing this register in the course of a normal CPU read cycle, bit 0-2 yields the register offset value (which is %*xxxxx*001).

The contents of the utility interrupt vector register must not be changed while one of the utility interrupts is active.

Bits 0-2 The lower three bits of the utility interrupt vector register are encoded by the VMEchip to uniquely identify the function that caused the utility interrupt request as shown in Table 4-7.

Bits 3-7 UVB3 through UVB7 are utility vector base bits.

The upper five bits of the register are programmable by software to provide a unique base for the vector provided in the course of acknowledging one of the utility interrupts. These bits are cleared by any reset.

Table 4-7. Encoding of the Interrupt ID

Utility Interrupt Source	**Bit 2**	**Bit 1**	**Bit 0**
SIGLP	0	0	1
LM0	0	1	0
IACK	0	1	1
LM1	1	0	0
SIGHP	1	0	1
SYSFAIL	1	1	0
WPBERR	1	1	1

Interrupt Request Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2015						IL2	IL1	IL0
						R/W	R/W	R/W

Note This register is used to configure the interrupt request line that the interrupter activates to request an interrupt on the VMEbus.

Bits 0-2 The three interrupt level select lines are encoded as shown in Table 4-8. Writing a non-0 value to these three bits causes the interrupter to activate the corresponding VMEbus IRQ line. Because the interrupter operates in the Release-On-Acknowledge (ROAK) mode, the interrupt request register is cleared, deactivating the IRQ line when the chip responds to a VMEbus interrupt acknowledge cycle. These bits are cleared by SYSRESET.

Table 4-8. Configuring the Interrupt Request Level

Interrupt Request Line Driven	IL2	IL1	IL0
None	0	0	0
IRQ1*	0	0	1
IRQ2*	0	1	0
IRQ3*	0	1	1
IRQ4*	1	0	0
IRQ5*	1	0	1
IRQ6*	1	1	0
IRQ7*	1	1	1

Note When the bits are set to drive one of the IRQ lines, they must not be changed. The three bits may be changed only when they are all cleared, signifying that the previous interrupt request has been serviced.

An added function provided by setting IACKEN (refer to the *Utility Interrupt Mask Register* section in this chapter) is provided by the VMEchip to signal the local processor when the interrupt request (generated through this register) has been acknowledged on the VMEbus.

VMEbus Status/ID Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2017	D07	D06	D05	D04	D03	D02	D01	D00
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Note This register allows software to program dynamically the status/ID that the interrupter provides during an interrupt acknowledge cycle. D00-D03 are set by SYSRESET, D04-D07 are cleared by SYSRESET.

Bus Error Status Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2019					RMCERR	VBERR	ACTO	LBTO
					R	R	R	R

Note This register allows the MC68030 to determine the cause of a bus error condition flagged by the VMEchip. Reading the register causes all of its bits to be cleared to 0. The bus error status register is designed to only indicate the cause of the latest bus error condition (i.e., when there is cause to set any of the bits, all other bits are cleared).

Bit 0	When set, this status bit indicates that the local timer has timed out.
Bit 1	When set, this status bit indicates that the VMEbus access timer has timed out.
Bit 2	When set, this status bit indicates that the VMEbus BERR* signal was activated in the course of a non write posted cycle that was initiated by the VMEchip. It should be noted that this bit is not set if the VMEbus global timer timed out in response to a VMEbus cycle that was initiated by another VMEbus master.
Bit 3	This bit should be ignored.

GCSR Base Address Configuration Register

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE201B					GCSRA7	GCSRA6	GCSRA5	GCSRA4
					R/W	R/W	R/W	R/W

Note This register allows software to set the base address of the GCSR set of registers in the VMEbus supervisor short I/O map.

The value contained in bits 0-3 of this register configures bits 4-7 of the GCSR base address. Address lines A08-A15 are fixed at $0. Refer to Table 4-9. Bits 1-3 of the VMEbus address select the specific registers in the GCSR. These bits are set to 1 by SYSRESET, therefore, unless otherwise programmed, the GCSR set does not respond to VMEbus accesses. GCSR functions are not enabled when the GCSR is mapped not to respond to VMEbus accesses. For example: location monitors SIGHP and SIGLP.

Table 4-9. VMEchip GCSR as Viewed from the VMEbus

GCSRA7-4	Short I/O Address of GCSR
$0	$0000-000F
$1	$0010-001F
$2	$0020-002F
$3	$0030-003F
$4	$0040-004F
$5	$0050-005F
$6	$0060-006F
$7	$0070-007F
$8	$0080-008F
$9	$0090-009F
$A	$00A0-00AF
$B	$00B0-00BF
$C	$00C0-00CF
$D	$00D0-00DF
$E	$00E0-00EF
$F	Does not respond

Programming the GCSR

There are eight GCSR registers as shown in Table 4-10. The VMEbus address is in the supervisor short I/O map.

Table 4-10. VMEchip Global Control and Status Register

MVME147 Address	VMEbus Address	Register								Name
		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
FFFE2021	00*x*1	LM3	LM2	LM1	LM0	CHIPID3	CHIPID2	CHIPID1	CHIPID0	Global 0
FFFE2023	00*x*3	R&H	SCON	ISF	BRDFAIL			SIGHP	SIGLP	Global 1
FFFE2025	00*x*5	BRDID7	BRDID6	BRDID5	BRDID4	BRDID3	BRDID2	BRDID1	BRDID0	Board ID
FFFE2027	00*x*7	General Purpose Control and Status Register 0								
FFFE2029	00*x*9	General Purpose Control and Status Register 1								
FFFE202B	00*x*B	General Purpose Control and Status Register 2								
FFFE202D	00*x*D	General Purpose Control and Status Register 3								
FFFE202F	00*x*F	General Purpose Control and Status Register 4								

Note The *x* denotes the value in the GCSR base address configuration register bits 0-3.

Note The bottom two lines in the table for each of the following GCSR register definitions defines the operations possible on the register bits, from the MC68030 and the VMEbus, as follows:

R	This bit is a read-only status bit.
R/W	This bit is readable and writable.
C	Writing a 1 to this bit clears it. This bit reads 0.
R/C	This bit is readable. Writing a 1 to this bit clears it.
R/S	This bit is readable. Writing a 1 to this bit sets it; it cannot be cleared.

Global Register 0

MVME147 ADDRESS	VMEbus ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2021	00X1	LM3	LM2	LM1	LM0	CHIPID3	CHIPID2	CHIPID1	CHIPID0
MC68030		R	R	R	R	R	R	R	R
VMEbus		R	R	R	R	R	R	R	R

Bits 0-3	These bits provide a unique identification number for the VMEchip. The VMEchip presents a hardwired ID of %0001.
Bit 4	Location monitor 0 is configured to monitor double-byte accesses to the supervisor short I/O address $00F0, and single-byte accesses to the short I/O address $00F1. When cleared, LM0 indicates that an access to address $00F0 or $00F1 was detected. At such a time, utility interrupt level 2 is requested (if the interrupt is enabled). LM0 is set when the interrupt is acknowledged or when software writes a 1 to it. This bit is set to 1 by SYSRESET. See Note below.
Bit 5	Location monitor 1 is configured to monitor double-byte accesses to the supervisor short I/O address $00F2, and single-byte accesses to the short I/O address $00F3. When cleared, LM1 indicates that an access to address $00F2 or $00F3 was detected. At such a time, utility interrupt level 4 is requested (if the interrupt is enabled). LM1 is set when the interrupt is acknowledged or when software writes a 1 to it. This bit is set to 1 by SYSRESET. See Note below.
Bit 6	Location monitor 2 is configured to monitor double-byte accesses to the supervisor short I/O address $00F4, and single-byte accesses to the short I/O address $00F5. When cleared, LM2 indicates that an access to address $00F4 or $00F5 was detected. LM2 is set when software writes a 1 to it. This bit is set to 1 by SYSRESET. See Note below.

Bit 7 Location monitor 3 is configured to monitor double-byte accesses to the supervisor short I/O address $00F6, and single-byte accesses to the short I/O address $00F7. When cleared, LM3 indicates that an access to address $00F6 or $00F7 was detected. LM3 is set when software writes a 1 to it. This bit is set to 1 by SYSRESET. See Note below.

Note The GCSR Base Address Configuration Register must be programmed to allow the GCSR set of registers to respond to VMEbus accesses for this function to be enabled.

The VMEbus master that executes the location monitor cycle must generate the DTACK signal to terminate the cycle.

Global Register 1

MVME147 ADDRESS	VMEbus ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2023	00X3	R&H	SCON	ISF	BRDFAIL			SIGHP	SIGLP
MC68030		R/W	R	R/W	R			R/C	R/C
VMEbus		R/W	R	R/W	R			R/S	R/S

Bit 0 The SIGLP control signal allows other VMEbus masters to interrupt the MC68030. SIGLP can only be set from the VMEbus. It can only be cleared by the MC68030. When a VMEbus master sets SIGLP to a 1, the VMEchip requests a level 1 interrupt to the MC68030 (if such interrupts are enabled). The interrupt request remains until the MC68030 writes a 1 to it. This bit is cleared by SYSRESET. See Note 1 below.

Bit 1 The SIGHP control signal allows other VMEbus masters to interrupt the MC68030. SIGHP can only be set from the VMEbus. It can only be cleared by the MC68030. When a VMEbus master sets SIGHP to a 1, the VMEchip requests a level 5 interrupt to the MC68030 (if such interrupts are enabled). The interrupt request remains until the MC68030 writes a 1 to it. This bit is cleared by SYSRESET. See Note 1 below.

Bit 4 BRDFAIL is a reflection of the BRDFAIL* input/output signal line. The status bit is set to 1 whenever the signal line is activated by either the VMEchip, or by a watchdog time-out from the PCC. The bit is cleared when the BRDFAIL* signal is deactivated.

Bit 5 The ISF control bit allows other VMEbus masters to cause the VMEchip to release its contribution to the VMEbus SYSFAIL* line. This is provided so that software can determine how many boards have failed. It should be noted that the ISF bit has no effect on the BRDFAIL status bit. Setting the bit to 1 inhibits the VMEchip from activating the VMEbus SYSFAIL* line. This bit is cleared by SYSRESET.

Bit 6	The SCON status bit is a reflection of the configuration of header J3. When J3 pins 1 and 2 are connected, enabling the MVME147 as system controller, the SCON bit is 1. Otherwise it is 0.
Bit 7	The R&H bit allows other VMEbus masters to reset the MVME147. The MVME147 is held in the reset state for as long as the R&H bit is set. This bit is cleared by SYSRESET.

Notes

1. The GCSR Base Address Configuration Register must be programmed to allow the GCSR set of registers to respond to VMEbus accesses for this function to be enabled.

2. If the MVME147 sets its own R&H, it causes itself to be maintained in a reset state until some other master clears the bit to 0.

3. Software must never activate R&H for shorter than 35 microseconds.

4. The R&H bit should not be set while the local MPU is executing a VMEbus cycle.

Board Identification Register

MVME147 ADDRESS	VMEbus ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2025	00X5	BRDID7	BRDID6	BRDID5	BRDID4	BRDID3	BRDID2	BRDID1	BRDID0
MC68030		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
VMEbus		R	R	R	R	R	R	R	R

Note The MC68030 can both read and write to this register. The VMEbus can only read it. This register allows the software to uniquely identify boards. The whole register is cleared by SYSRESET.

General Purpose CSR 0

MVME147 ADDRESS	VMEbus ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2027	00X7	General Purpose Control and Status Register 0							
MC68030		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
VMEbus		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Note General purpose CSR 0 is both readable and writable from the MC68030 and from the VMEbus. All of its bits are set to 1 at SYSRESET.

General Purpose CSR 1-4

MVME147 ADDRESS	VMEbus ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
FFFE2029	00X9	General Purpose Control and Status Register 1							
FFFE202B	00XB	General Purpose Control and Status Register 2							
FFFE202D	00XD	General Purpose Control and Status Register 3							
FFFE202F	00XF	General Purpose Control and Status Register 4							
MC68030		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
VMEbus		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Note General purpose CSR 1-4 are both readable and writable from the MC68030 and from the VMEbus. All of their bits are cleared to 0 at SYSRESET.

Functional Description 5

Introduction

This chapter provides a functional description of the MVME147. The functional description provides an overview of the module, followed by a detailed description of each section of the module. The block diagram of the MVME147 is shown in Figure 5-1.

Functional Description

The MVME147 is a complete microcomputer system. The module contains the following:

- ❏ MC68030 MPU
- ❏ 4MB or more of DRAM (accessible from the VMEbus)
- ❏ MC68882 Floating-Point Coprocessor
- ❏ VMEchip
- ❏ 4088 bytes of static RAM with battery backup
- ❏ Time-of-day clock with battery backup
- ❏ Four serial ports with EIA-232-D interface,
- ❏ Two tick timers
- ❏ Watchdog timer
- ❏ Four ROM sockets
- ❏ SCSI bus interface with DMA
- ❏ Ethernet transceiver interface
- ❏ Centronics printer port
- ❏ A32/D32 VMEbus interface
- ❏ VMEbus system controller
- ❏ Numerous control functions

Note that the Ethernet interface is not included on the MVME147-010.

MC68030 MPU

The MC68030 is the main processor of the MVME147. The MC68030 has onchip instruction and data caches. The MVME147 prevents the MC68030 from caching accesses to any other device than local DRAM by activating the cache inhibit in pin (CIIN*) during the accesses. Refer to the *MC68030 Enhanced 32-Bit Microprocessor User's Manual*.

The MC68030 includes a software reset instruction; however, the MVME147 does not support this instruction.

MC68882 FPC

The MC68882 Floating-Point Coprocessor (FPC) is connected to the MC68030 as a 32-bit port. It runs at the same frequency as the MC68030. Refer to the *MC68881/MC68882 Floating-Point Coprocessor User's Manual* for a detailed description of its operation.

VMEchip

The VMEchip is an Application Specific IC (ASIC) device designed to reduce the real estate required to interface with the VMEbus. It provides the following:

- ❏ VMEbus system controller functions
- ❏ VMEbus interrupt handler
- ❏ VMEbus and local time-out functions
- ❏ MC68030 to VMEbus interface
- ❏ VMEbus control signal drivers and receivers.

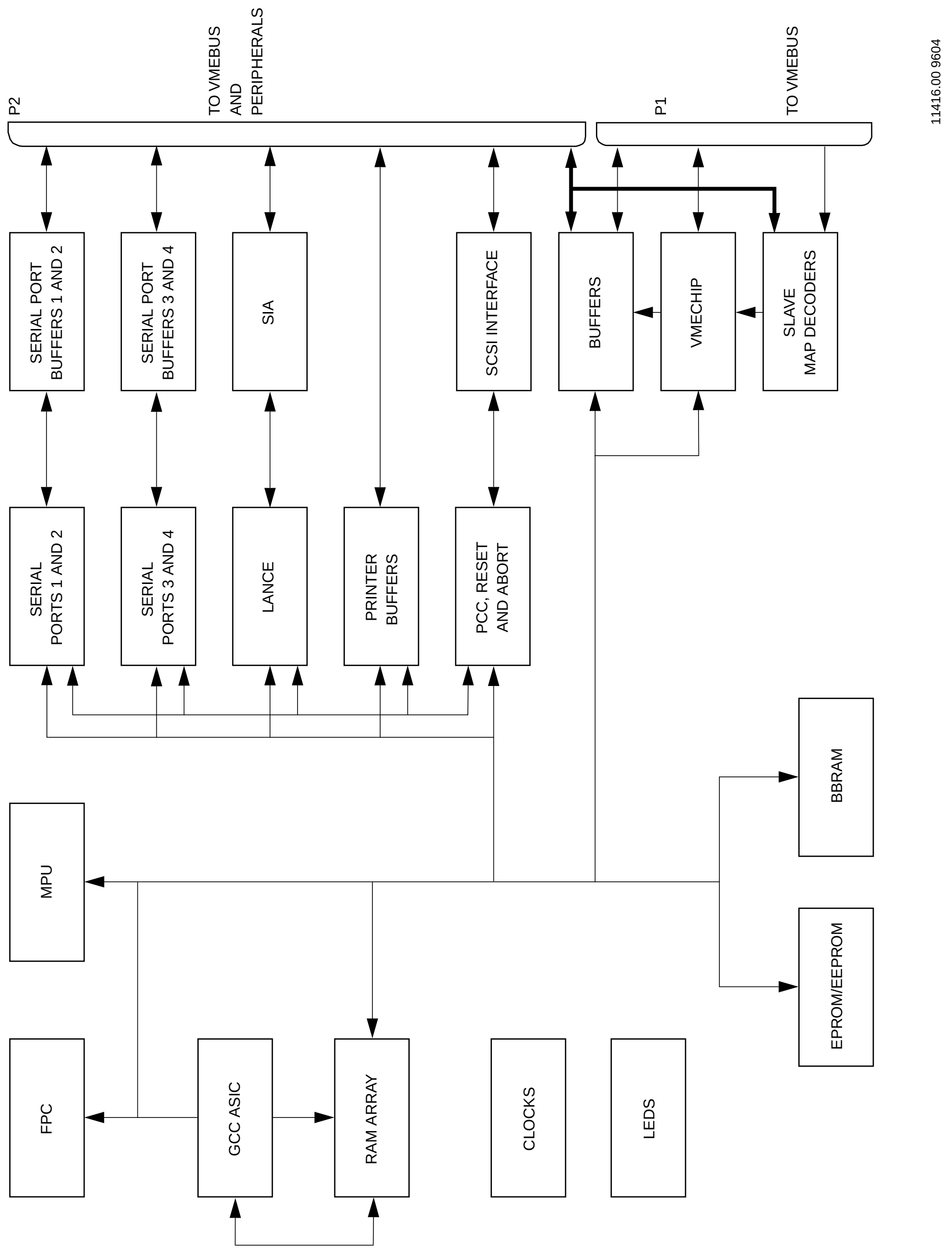


Figure 5-1. MVME147 Block Diagram

VMEbus System Controller

One of the many functions provided by the VMEchip is the VMEbus system controller function. The system controller includes the following:

- ❏ VMEbus global time-out timer
- ❏ System Clock (SYSCLK*) driver
- ❏ Arbiter
- ❏ Interrupt Acknowledge (IACK*) daisy-chain driver.

The system reset utility is also described here because it is enabled when the MVME147 is system controller. The system controller function is enabled/disabled by header J3. When the MVME147 is system controller, the System Controller (SCON) LED is turned on.

VMEbus Time-Out

The VMEbus timer is started when either Data Strobe (DS0* or DS1*) goes active and is disabled when they both go inactive. If the timer times out before the data strobes go inactive, the Bus Error (BERR*) signal is activated. The time-out period is controlled by the timer interval register and may be 102 µs, 205 µs, 410 µs, or infinite.

System Clock Utility

The 16 MHz system clock is driven onto the VMEbus SYSCLK* signal line by the VMEchip system clock driver.

Arbiter

The VMEchip implements two different arbitration modes. They are prioritized and round-robin. The mode is software selectable.

In the prioritized mode, the arbiter prioritizes the bus request signals and responds with grant to the highest priority requester. The arbiter also informs the current bus master by activating the Bus Clear (BCLR*) signal when a request from a higher priority master has been received.

In the round-robin mode, the arbiter assigns the bus on a rotating priority basis. The BCLR* signal is not used in the round-robin mode.

The arbiter also contains a time-out feature. It activates Bus Busy (BBSY*) on its own if BBSY* is not activated by the requester within the time-out period. The time-out period is software selectable and may be set to 410 µs or infinite.

IACK* Daisy-Chain Driver

The IACK* daisy-chain driver activates the interrupt acknowledge daisy-chain whenever an interrupt handler acknowledges an interrupt request.

System Reset Function (SYSRESET*)

Even though SYSRESET* is not a VMEbus system controller function, the MVME147 enables/disables its SYSRESET* function at the same time that it enables/disables its system controller functions. When the MVME147 is system controller, it drives the SYSRESET* signal line whenever an onboard reset is generated (it does not fully implement the SYSRESET* timing of a VMEbus power monitor).

VMEbus Interrupter

The VMEchip incorporates a flexible, multilevel bus interrupter module. This module can activate an interrupt on the VMEbus at any of the seven interrupt levels.

The VMEchip interrupter also monitors the VMEbus to determine when an interrupt acknowledge cycle is in process. When the VMEchip receives an interrupt-acknowledge-in signal from the VMEbus and it is currently interrupting at the acknowledge level, it responds with a status/ID vector. Otherwise, it generates an interrupt-acknowledge-out signal to the VMEbus. The VMEchip is a 8-bit interrupter and consequently responds to all sizes of interrupt acknowledge cycles.

Local Bus Time-Out

The VMEchip provides a time-out function for the MC68030 local bus. When the timer times out, a bus error signal is sent to the MC68030. The time-out value is selectable in software for 102 µs, 205 µs, 410 µs, or infinite.

Note The local bus timer does not operate during VMEbus bound cycles. VMEbus bound cycles are timed by the VMEbus access timer and by the global timer (not necessarily on the module).

VMEbus Access Time-Out

The VMEchip provides a VMEbus access time-out timer. If the MVME147 is not granted the VMEbus within the selected time period, the MC68030 receives a bus error signal (unless the cycle is write posted). The time period is selectable in software for 102 µs, 1.6 ms, 51 ms, or infinite.

VMEbus Master Interface

The VMEbus master interface is provided by the VMEchip. Depending on the VMEbus address, the MVME147 master interface may be A32/D32, A24/D16, or A16/D16. When the MC68030 needs the VMEbus for a read, write, read-modify- write, or interrupt acknowledge cycle, it requests the VMEchip to obtain bus mastership. The VMEchip requests the bus and after it receives mastership, it activates the VMEbus signals as requested by the MC68030. When the slave responds, the VMEchip passes this information to the MC68030.

VMEbus Requester

The VMEbus requester is used to obtain and relinquish mastership of the VMEbus. Its operation is affected by software programmable bits in the VMEchip.

The requester requests VMEbus mastership at the programmed level when the board is not the current VMEbus master and one of the following happens:

- ❏ The MC68030 executes a program space cycle that is bound for the VMEbus.
- ❏ The MC68030 executes a data space cycle that is bound for the VMEbus.
- ❏ The MC68030 executes an IACK cycle that is bound for the VMEbus.
- ❏ The MC68030 sets the DWB bit in the VMEchip.
- ❏ The MC68030 executes a “multiple address RMC” cycle that is bound for the local DRAM and the WAITRMC bit is set in the PCC.

Requesting VMEbus mastership is also affected by the RONR bit in the VMEchip LCSR.

The requester maintains VMEbus mastership as long as one of the following conditions is met:

- ❑ The MC68030 is executing a VMEbus cycle.
- ❑ The RWD bit is cleared in the VMEchip and no other VMEbus master is activating a bus request.
- ❑ The RNEVER bit is set in the VMEchip.
- ❑ The DWB bit is set in the VMEchip.
- ❑ The MC68030 is performing an RMC sequence to the VMEbus.
- ❑ The MC68030 is finishing an RMC sequence that started in local DRAM while the WAITRMC bit was set in the PCC.

VMEbus Slave Interface

The VMEchip provides the VMEbus slave interface for the MVME147. When the VMEbus wants to access the DRAM or VMEchip control registers, the VMEbus map decoder selects the VMEchip.

For a DRAM access, the VMEchip requests the local bus and after obtaining mastership, the VMEchip activates the proper signals to access the DRAM. The DRAM notifies the VMEchip and the VMEchip notifies the VMEbus when the DRAM has completed.

Mastership of the local bus is not required to access the VMEchip Global Control and Status Registers. The VMEbus cannot access any other resources than DRAM on the MVME147.

General Control Chip (GCC)

The GCC is a CMOS ASIC designed to replace 16 PALs and 19 other devices on the previous MVME147 single board computers. It provides the interface between the MC68030, VMEchip, PCC, LANCE, and DRAM on the MVME147. While preserving the original function of the MVME147, the GCC lowers part and manufacture cost and reduces power consumption.

The GCC includes the following features:

- ❑ Local DRAM controller:
  - Control signals to DRAM array
  - MPU to DRAM interface
  - PCC to DRAM interface
  - LANCE to DRAM interface
  - VMEchip to DRAM interface
  - Local bus map decoder for MPU and PCC to DRAM
  - Refresh controller
  - Byte parity generator/checker
- ❑ External selectable internal/external local bus arbiter
- ❑ MC68030 to MC68882 interface logic
- ❑ Local bus BERR* generation from parity and other errors
- ❑ Support for WAITRMC function
- ❑ MC68030 to VMEbus map decoder
- ❑ Other miscellaneous control logic: CIIN*, SCIRST*, ASYNCH*

SCSI Reset

Because the WD33C93 does not implement SCSI bus Reset (RST), the GCC has separate signals to sense and drive it.

RAM Refresh Timer

The DRAM used on the MVME147 must be refreshed at least every 15.6 µs. The GCC provides a refresh signal to the DRAM at least once every 15.6 µs.

Local Bus Multiport Arbiter

Because the local address and data buses are used to access the onboard DRAM and the VMEbus, any devices that use these resources must become the local bus master first. The MC68030 arbitration logic (Bus Request (BR*), Bus Grant (BG*), Bus Grant Acknowledge (BGACK*)) is used by the GCC multiport arbiter to transfer local bus mastership from the current master to the next master. During normal operation the MC68030 is the local bus master.

When the PCC, the LANCE, or the VMEbus requests use of the local bus, the GCC multiport arbiter activates BR* to the MC68030. The MC68030 responds by activating BG*, finishing its current cycle (if one is in progress), and giving up local bus mastership.

At this point, the GCC multiport arbiter grants local bus mastership to the highest priority requesting device. The granted device uses the local bus and then relinquishes local bus mastership. If another device is requesting local bus mastership at this time, the GCC multiport arbiter grants it to the device; otherwise the MPU resumes local bus mastership.

The arbitration priority in high to low order is: LANCE, PCC, VMEbus, and MPU.

Duplication of PCC Functions

Several registers and functions that exist in the PCC have been duplicated in the GCC. Access to these registers and functions is still through the PCC addresses. However, some pins on the PCC are now unconnected because the logic is inside the GCC.

Peripheral Channel Controller (PCC)

The PCC is an ASIC device designed for the MVME147. The PCC includes the following features:

- DMA channel for SCSI data
- 8-bit to 32-bit converter for SCSI data
- SCSI chip interface
- Local processor interrupter/handler
- Peripheral chip map decoder
- Two programmable tick timers
- Watchdog timer
- Parallel (Centronics) printer interface
- Control and status registers
- RESET and ABORT switch interface
- Power-up reset interface
- AC Fail interrupter
- Refresh timer for local DRAM (function duplicated in GCC; PCC function not used)

DMA Channel Controller (DMAC)

The PCC includes a DMA Channel Controller (DMAC) to move data between the SCSI chip and memory. The DMA channel features a 32-bit address pointer for data transfers, a 32-bit pointer for the command chaining table, and a 24-bit byte counter.

Because of its 8-bit to 32-bit data bus width conversion, the chip moves SCSI data at rates up to 1.5MB/second while using less than 25 percent of the local bus bandwidth when doing a DMA to local DRAM.

DMAC Initiation Mode

The DMAC has two initiation modes: direct and command chaining (scatter-gather).

In the direct mode, the data address pointer and the byte count are loaded into the chip.

In the command chaining mode, a table of data addresses and byte counts is placed in local RAM and the address of the table is loaded into the chip. The chip walks through the addresses and byte counts from the local RAM to move each block of data as indicated by the table. Scatter-gather operations are supported by command chaining.

The PCC can DMA to/from local DRAM and VMEbus memory only. Any other access results in a local bus time-out.

DMAC Operation States

The DMAC is always in one of three operational states: idle state, table walk state, or data transfer state. The DMA sequences through the three depending upon the contents of the DMA control register which is initialized by the MPU.

Idle State

The DMAC starts out from reset in the idle state. It stays in the idle state until the DMAC is enabled (DMAEN set to 1). It returns to the idle state when the DMAC has completed the requested operations (normally or with error). It does not leave the idle state again until all error status bits are cleared and DMAEN is again set to 1.

Data Transfer State

When DMAEN is set, the DMAC goes directly to the data transfer state unless the Table Walk (TW) bit is set. If TW is set, the DMAC table walks before entering the data transfer state.

In either event, when the data transfer state is entered, the DMAC moves data between local DRAM and the WD33C93 (SCSI bus interface controller). The DMAC reads/writes data in local DRAM using the address contained in the data address register. Data transfers continue until the byte count register reaches 0.

At this point, the DMAC sets the done bit and enters the idle state unless more table walking is indicated by the link bit in the byte count register.

Table Walk State

The table address and table function code registers point to the DMAC table. Table 5-1 is an example. The table has two entries for each data block: the data address and the byte count.

When the DMAC table walks, it copies the first longword from the table into the table address register, and the second longword from the table into the byte count register. It then goes to the data transfer state. If the table walk caused the link bit to be set, the DMAC table walks again after the data transfer state has ended.

Note The DMAC table must always be placed within 32-bit memory. The PCC terminates if 8-bit or 16-bit memory is encountered during a table walk.

Table 5-1. Example DMAC Table

Memory Address	Data	Comments
$00010000	$00020000	The first block of data starts at $20000
$00010004	$85000100	There are more entries, FCs = 5, move $100 bytes
$00010008	$00128000	The second block of data starts at $128000
$0001000C	$83001000	There are more entries, FCs = 3, move $1000 bytes
$00010010	$00045000	The third block of data starts at $45000
$00010014	$03000050	There are no more entries, FCs = 3, move $50 bytes

DMAC Error Conditions

If any error is encountered during the table walk or the data transfer state, the DMAC goes immediately to the idle state. In addition, it sets the done bit and the appropriate error bit.

SCSI Data Bus Converter

The WD33C93 connects to a separate 8-bit data bus on the PCC and not to the local MC68030 bus. This allows the PCC to collect one longword of data by transferring one byte at a time from the WD33C93 without using the processor bus. When a longword is ready, the chip requests the local bus and transfers it. This scheme lightens the load on the MC68030 local bus.

SCSI Chip Interface

The PCC provides the interface for MC68030 accesses of the WD33C93. It uses the nonmultiplexed mode which requires that the software use the WD33C93 pointer registers to access its internal registers. The WD33C93 registers are accessible indirectly through the address register at $FFFE4000.

Programmable Tick Timers

The PCC features two 16-bit programmable tick timers. A timer generates a periodic interrupt to the MC68030 at the programmed rate. The period is 6.25 μs to 0.4 seconds in 6.25 μs increments. The timer may also be disabled. The timer interrupt level is programmable and it provides a status/ID vector when its interrupt is acknowledged.

Watchdog Timer

The PCC includes a watchdog timer function. When enabled by software, the watchdog timer may be programmed to reset the module if it is system controller. Whenever the watchdog timer times out, the FAIL LED is lit (in addition to when the BRDFAIL bit is set in the VMEchip).

The watchdog timer counts outputs from tick timer 1. If the watchdog timer is not reset by software within the programmed number of ticks, it times out.

Printer Interface

The PCC has a Centronics compatible printer interface. The printer interface interrupts the MC68030 when it is ready for data or when a fault occurs. The interrupt level is programmable and it provides a status/ID vector when requested.

Control and Status Registers

The PCC has input and output signal lines for controlling various functions on the MVME147. There are control lines for DRAM parity enable, parity test and parity error status, VMEbus map select, multiple address RMC mode, and LANCE address select.

RESET and ABORT Switches

The PCC provides the RESET and ABORT switch interface.

The RESET switch signal is debounced and when it is enabled, it causes a reset out signal. The RESET switch can be enabled/disabled by software.

The ABORT switch signal is debounced and sent to the level 7 interrupter. When it is enabled, the ABORT switch causes a level 7 interrupt to the MC68030. The interrupter returns a status/ID vector when requested. The ABORT switch can be enabled/disabled by software.

Power-Up Reset

When the PCC receives a power-up reset signal, it generates a reset out signal and sets the power-up bit in the control register. The power-up bit can be used by software to determine when a power-up reset has occurred.

AC Fail Interrupter

When the AC Fail interrupt is enabled and the PCC receives an AC Fail signal, a Level 7 interrupt is sent to the MC68030. The AC Fail interrupt can be enabled/disabled by software. The AC Fail interrupter provides a status/ID vector when requested.

Serial Port Interface

The MVME147 uses two Z8530 Serial Communications Controller (SCC) devices to implement the four serial ports. A 5 MHz clock is used to generate the baud rate clock and the serial ports support the standard baud rates (110 through 19,200). Serial port 4 also supports synchronous modes of operation.

The four serial ports on the MVME147 are different functionally because of the limited number of pins on the P2 connector. Serial port 1 is a minimum function asynchronous port. It uses RXD, CTS, TXD, and RTS. Serial ports 2 and 3 are full function asynchronous ports. They use RXD, CTS, DCD, TXD, RTS, and DTR. Serial port 4 is a full function asynchronous or synchronous port. It uses RXD, CTS, DCD, TXD, RTS, and DTR. It also interfaces to the synchronous clock signal lines.

All four serial ports use EIA-232-D drivers and receivers located on the MVME147 and all the signal lines are routed to P2. The configuration headers are located on the MVME147 and the MVME712. An external I/O transition board such as the MVME712 must be used to convert the P2 pin out to industry standard connectors.

Headers on the MVME712 provide jumper selectable options for each EIA-232-D interface to be configured to connect to DTE or DCE.

For additional information on the SCCs, refer to the Zilog literature listed in *Related Documents*.

Ethernet Interface

The Ethernet interface is not used on the MVME147-010.

The MVME147 uses the AM79C90 Local Area Network Controller for Ethernet (LANCE) and the AM7992 Serial Interface Adapter (SIA) to implement the Ethernet transceiver interface. The balanced differential transceiver signal lines from the AM7992 are coupled via an onboard transformer to signal lines that go to the P2 connector and eventually to the MVME712 transition board, where they are connected to an industry standard DB-15 connector.

The AM79C90 performs DMA operations to perform its normal functions. The MVME147 restricts AM79C90 DMA to local DRAM only. The AM79C90 cannot access the VMEbus. If the DRAM size is less than 16MB then it repeats itself in the AM79C90 16MB memory map. If it is more than 16MB, then the AM79C90 accesses the section of DRAM defined by the LANA24 and LANA25 bits in the PCC Slave Base Address Register (bits 6 and 7 of $FFFE102B).

Every MVME147 is assigned an Ethernet station address. The address is $08003E2*xxxxx* where *xxxxx* is the unique number assigned to the module (i.e., every MVME147 has a different value for *xxxxx*).

Each Ethernet station address is displayed on a label attached to the MVME147's backplane connector P2. In addition, the *xxxxx* portion of the Ethernet station address is stored in BBRAM, location $FFFE0778 as $2*xxxxx*.

If Motorola networking software is running on an MVME147, it uses the 2*xxxxx* value from BBRAM to complete the Ethernet station address ($08003E2*xxxxx*). The user must assure that the value of 2*xxxxx* is maintained in BBRAM. If the value of 2*xxxxx* is lost in BBRAM, the user should use the number on the label on the P2 connector to restore it. Note that MVME147bug includes the "LSAD" command for examining and updating the BBRAM *xxxxx* value.

If non-Motorola networking software is running on an MVME147, it must set up the AM79C90 so that the Ethernet station address is that shown on the label to ensure that the module has a globally unique Ethernet station address.

SCSI Interface

The MVME147 has a SCSI mass storage bus interface. The SCSI bus is provided to allow mass storage subsystems to be connected to the MVME147. These subsystems may include hard and floppy disk drives, streaming tape drives, and other mass storage devices.

The SCSI interface is implemented using the WD33C93 controller. DMA to/from the WD33C93 is implemented through the PCC.

Data Bus Structure

The data bus structure on the MVME147 is arranged to accommodate the 8-bit, 16-bit, 32-bit, and 16/32-bit ports that reside on the board. The 8-bit ports are connected to D24-D32 of the local bus, 16-bit ports are connected to D16- D32 of the local bus and 32-bit ports are connected to D00-D32 of the local bus.

Battery Backed Up RAM and Clock

The SGS-Thompson M48T18 RAM and clock chip is used on the MVME147. This chip provides a time-of-day clock, oscillator, power fail detection, memory write protection, and 8184 bytes of RAM. However, only 4088 bytes of RAM are accessible on the MVME147. The battery and crystal plug into the M48T18.

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.

Note Versions of the MVME147 that use the MK48T02 only have 2040 bytes of RAM available. Any programs written to use RAM in the address range above the TOD clock will not work on an earlier version MVME147.

The internal battery has a typical life span of 3 to 5 years when the clock is running and a minimum of 10 years when the clock has not been put in operation.

Note The clock is shipped from the factory in the power-save mode.

ROM/PROM/EPROM/EEPROM

There are four 32-pin ROM/PROM/EPROM/EEPROM sockets on the MVME147. They are organized as 2 banks with two sockets per bank.

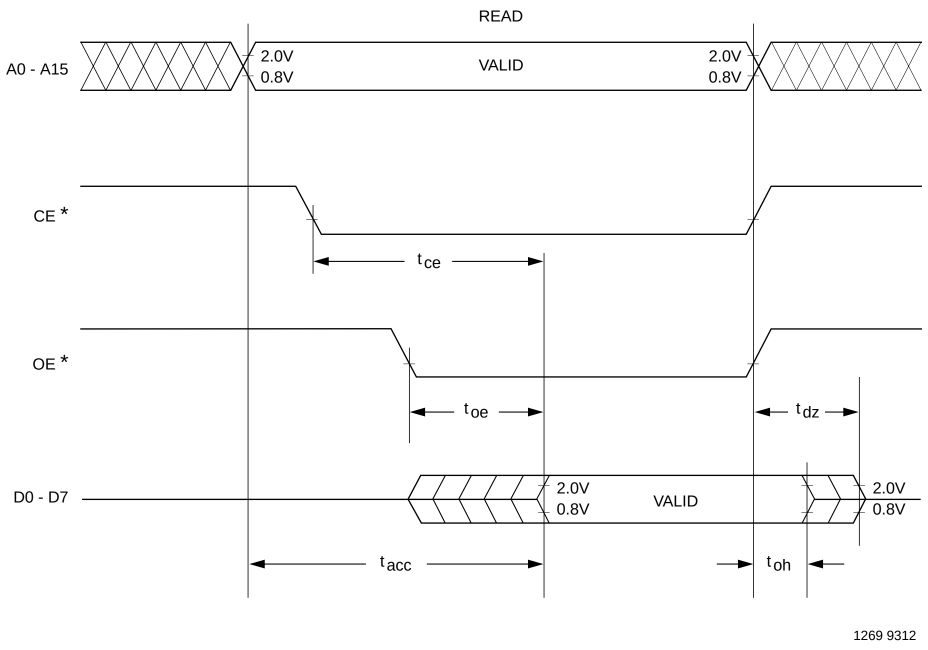
The banks are configured as word ports to the MPU. Each bank can be separately configured for 8K x 8, 16K x 8, 32K x 8, 64K x 8, 128K x 8, 256K x 8, 512K x 8, or 1M x 8 ROM/PROM/EPROM devices or 2K x 8, 8K x 8, or 32K x 8 EEPROM devices.

There are several different algorithms for erasing/writing to EEPROM devices depending on the manufacturer. The MVME147 supports only those devices which have a static RAM compatible erase/write mechanism such as the Xicor X28256 or X2864H.

Device Timing Requirements

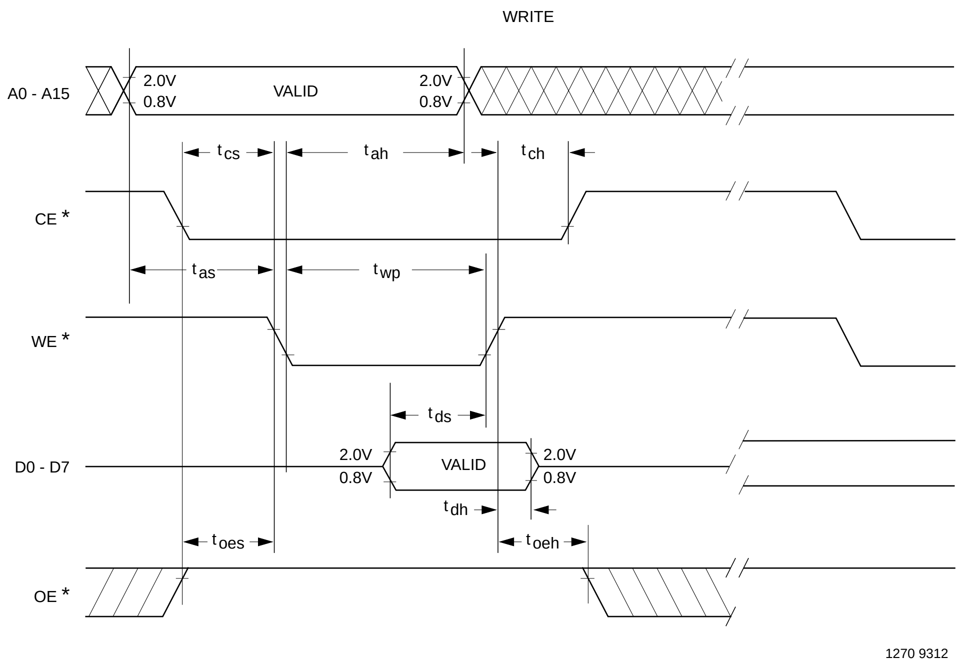
The ROM/PROM/EPROM/EEPROM devices must meet the timings shown in Figure 5-2.

The ROM/PROM/EPROM/ EEPROM devices are guaranteed the timings shown in Figure 5-3.



Symbol	Description	Minimum	Maximum	Unit
t_{acc}	Address valid to data valid		250	ns
t_{ce}	CE* low to data valid		250	ns
t_{oe}	OE* low to data valid		200	ns
t_{oh}	Address invalid, CE* or OE* high to data not valid	0		
t_{dz}	CE* or OE* high to data high impedance		100	ns

Figure 5-2. Timings Required by the MVME147



Symbol	Description	Minimum	Maximum	Unit
t_{as}	Address valid to WE* low	50		ns
t_{cs}	CE* low to WE* low	50		ns
t_{oes}	OE* high to WE* low	70		ns
t_{ah}	Address invalid after WE* low	200		ns
t_{wp}	WE* low pulse width	190		ns
t_{ds}	Data valid to WE* high	160		ns
t_{dh}	WE* high to data not valid	5		ns
t_{oeh}	WE* high to OE* low	100		ns
t_{ch}	WE* high to CE* high	10		ns

Figure 5-3. Timings Guaranteed by MVME147

EEPROM Power-Up/Power-Down Considerations

The MVME147 provides no protection against inadvertent writes to EEPROM that might happen at power-up or power-down time. Most devices provide some level of internal protection. To gain "absolute protection" devices with additional "software protection" are recommended.

Interrupt Handler

The MC68030 may be interrupted by many sources. All interrupt sources are software enabled/disabled. Some have software programmable levels and all interrupt sources supply a vector during an interrupt acknowledge cycle.

The PCC chip decodes the MC68030 address bus and function codes to determine when an interrupt cycle is in progress. When the PCC detects an interrupt acknowledge cycle at the level it is interrupting on, it passes a status/ID vector. Otherwise, it generates an interrupt acknowledge out signal to the VMEchip.

When the VMEchip detects an interrupt acknowledge in signal from the PCC and it is interrupting at that level, it passes a status/ID vector. Otherwise, it requests mastership of the VMEbus (if it does not have mastership), and it drives the VMEbus signal lines to initiate an interrupt acknowledge cycle. The interrupting slave returns a status/ID vector.

Within a level, the interrupts from the PCC have the highest priority followed by the VMEchip and the VMEbus interrupts have the lowest priority.

Interrupt sources and vectors are listed in Table 5-2.

Table 5-2. MVME147 Interrupt Sources and Vectors

Interrupt Source	Path	Vector Source	Vector	Level
ACFAIL	PCC	PCC	%*xxxx*0000	7
BERR	PCC	PCC	%*xxxx*0001	7
ABORT	PCC	PCC	%*xxxx*0010	7
Serial Ports	PCC	Z8530 devices	See Z8530 data sheet	Prog
	PCC	PCC	%*xxxx*0011	Prog
LANCE	PCC	PCC	%*xxxx*0100	Prog
SCSI Port	PCC	PCC	%*xxxx*0101	Prog
SCSI DMA	PCC	PCC	%*xxxx*0110	Prog
Printer Port	PCC	PCC	%*xxxx*0111	Prog
Tick Timer 1	PCC	PCC	%*xxxx*1000	Prog
Tick Timer 2	PCC	PCC	%*xxxx*1001	Prog
Software Int. 1	PCC	PCC	%*xxxx*1010	Prog
Software Int. 2	PCC	PCC	%*xxxx*1011	Prog
WPBERR	VMEchip	VMEchip	%*yyyyy*111	7
SYSFAIL	VMEchip	VMEchip	%*yyyyy*110	6
SIGHP	VMEchip	VMEchip	%*yyyyy*101	5
LM1	VMEchip	VMEchip	%*yyyyy*100	4
IACK	VMEchip	VMEchip	%*yyyyy*011	3
LM0	VMEchip	VMEchip	%*yyyyy*010	2
SIGLP	VMEchip	VMEchip	%*yyyyy*001	1
VMEbus IRQ7*	VMEchip	From interrupting VMEbus slave	Determined by VMEbus slave	7
VMEbus IRQ6*	VMEchip	Same as above	Same as above	6
VMEbus IRQ5*	VMEchip	Same as above	Same as above	5
VMEbus IRQ4*	VMEchip	Same as above	Same as above	4

Table 5-2. MVME147 Interrupt Sources and Vectors (Continued)

Interrupt Source	Path	Vector Source	Vector	Level
VMEbus IRQ3*	VMEchip	Same as above	Same as above	3
VMEbus IRQ2*	VMEchip	Same as above	Same as above	2
VMEbus IRQ1*	VMEchip	Same as above	Same as above	1

NOTES: 1. *xxxx* is the value programmed into the PCC interrupt base vector register (address $FFFE102D) bits 4 through 7.

2. *yyyyy* is the value programmed into the VMEchip utility interrupt vector register (address $FFFE2013) bits 3 through 7.

Front Panel Switches and Indicators

There are two switches on the front panel of the MVME147. The switches are RESET and ABORT.

The RESET switch resets all onboard devices and drives SYSRESET* if the MVME147 is the system controller. The RESET switch may be disabled by software. Refer to the *RESET and ABORT Switches* section in this chapter.

The ABORT switch generates a Level 7 interrupt when enabled. It is normally used to abort program execution and return to the debugger. The ABORT switch may be disabled by software. Refer to the *RESET and ABORT Switches* section in this chapter.

There are four LED indicators on the front panel of the MVME147. The indicators are RUN, STATUS, FAIL, and SCON.

- ❑ RUN is lit when the MC68030 Address Strobe (AS*) pin is low.
- ❑ STATUS is lit when the MC68030 STATUS* pin is low.
- ❑ FAIL is lit when the Board Fail (BRDFAIL) bit is set in the VMEchip or when watchdog time-out occurs in the PCC.
- ❑ SCON is lit when the MVME147 is the VMEbus system controller (selected by jumper J3).

Onboard DRAM

The DRAM is accessible by the MC68030, PCC, LANCE, and VMEbus. It is specifically optimized for the MC68030.

The parity feature is not implemented on the MVME147-010.

The MVME147 has parity check which operates in one of three user selectable modes.

In mode 1, no parity checking is performed and the DRAM operates at maximum speed.

In mode 2, parity checking is performed for all bus masters and the DRAM operates at maximum speed when the MC68030 is bus master. When a parity error occurs in mode 2 and the MC68030 is the local bus master, the bus error signal is not activated during the current cycle. The bus error is activated during all subsequent MC68030 DRAM cycles. All other bus masters are notified of parity errors during the current cycle, consequently their DRAM access time increases by 1 clock.

In mode 3, parity checking is performed for all bus masters and parity errors are reported during the current cycle. In this mode, the DRAM access time is extended by one clock cycle to allow for parity checking.

MC68030 DRAM Accesses

The MC68030 is the default local bus master, therefore it is the local bus master as long as no other device requests local bus mastership.

PCC DRAM Accesses

When the PCC needs to transfer data, it requests local bus mastership from the GCC multiport arbiter. When the PCC has been granted local bus mastership, it executes one bus cycle and then releases bus mastership. If a parity error is detected during a PCC to DRAM read cycle, a bus error is returned to the PCC.

VMEbus DRAM Accesses

When the VMEbus map decoder detects an onboard DRAM select, the VMEchip requests local bus mastership from the GCC multiport arbiter. When the GCC multiport arbiter has granted local bus mastership, a DRAM read or write cycle happens and the VMEchip activates the DTACK* (or BERR* if parity is enabled and a parity error occurs) signal on the VMEbus.

If the VMEbus master is executing a read-modify-write cycle (RMC) to the DRAM, the GCC multiport arbiter allows re-arbitration of the local bus between the read and write portions of the sequence. It does not, however, allow the MC68030 to regain local bus mastership until both the read and write cycles have occurred to the DRAM.

When the VMEbus requests local bus mastership and the MC68030 is the current local bus master and it is executing a cycle that requires the VMEbus, then a dual port lockup condition occurs and the VMEchip signals a retry to the MC68030 by activating the BERR* and HALT* signal lines together. The MC68030 responds by aborting the current cycle, at which time it relinquishes local bus mastership so that the GCC multiport arbiter can grant it to the VMEbus. When the VMEbus has finished with the DRAM, the GCC multiport arbiter returns local bus mastership to the MC68030 and it retries the cycle that was aborted to allow the dual port access.

LANCE DRAM Accesses

When the LANCE needs to access DRAM it requests local bus mastership from the GCC multiport arbiter. When granted, the LANCE performs up to 16 DRAM accesses, then gives up local bus mastership. If a parity error occurs while enabled, the DRAM controller indicates it by not activating LANRDY* to the LANCE. The LANCE sees this as a memory fault and gives up local bus mastership.

Refresh

The DRAM devices require that each of their 4096/2048/1024 rows be refreshed once every 64/32/16 milliseconds. To accomplish this, once every 15 microseconds, the refresh timer requests that the RAM sequencer perform a Column Address Strobe (CAS) before Row Address Strobe (RAS) refresh cycle.

Reset

There are five sources of reset on the MVME147:

- SYSRESET* -- Resets all onboard devices.
- Power on reset -- Resets all onboard devices and drives SYSRESET* if this board is system controller.
- Front panel RESET -- Resets all onboard devices and drives SYSRESET* if this board is system controller.
- Remote reset -- When a remote switch is connected to front panel connector J4, it functions the same as the front panel RESET switch.
- Watchdog time-out -- Resets all onboard devices and drives SYSRESET* if this board is system controller.
- MC68030 RESET instruction -- Does nothing.

Sources of Bus Error (BERR*)

The devices on the MVME147 that are capable of activating a local bus error are described below.

Local Bus Time-Out

A Local Bus Time-Out (LBTO) occurs whenever an MPU or PCC access (outside of the VMEbus range) does not complete within the programmed time. If the system is configured properly, this should only happen if software accesses a nonexistent location within the onboard address range. Whenever an LBTO occurs, the LBTO status bit is set in the VMEchip.

VMEbus Access Time-Out

A VMEbus Access Time-Out (VATO) occurs whenever a PCC or MC68030 VMEbus bound cycle does not receive a VMEbus Bus Grant within the programmed time. This is usually caused by another bus master holding the bus for an excessive period of time. When a VATO occurs, the VATO status bit is set in the VMEchip.

VMEbus BERR*

The VMEbus BERR* occurs when the BERR* signal line is activated on the VMEbus while the MC68030 or PCC is the VMEbus master. VMEbus BERR* should occur only if:

- ❏ An initialization routine samples to see if a device is present on the VMEbus and it is not.
- ❏ Bad software accesses a nonexistent device within the VMEbus range.
- ❏ Bad configuration tries to access a device on the VMEbus incorrectly (such as driving LWORD* low to a 16-bit board).
- ❏ A hardware error occurs on the VMEbus.

- A VMEbus slave reports an access error (such as parity error).
- Whenever a VMEbus BERR* occurs, the VMEbus BERR* status bit is set in the VMEchip.

Local RAM Parity Error

When parity checking is enabled, the current bus master receives a bus error (or no LANRDY*, if LANCE) if it is accessing the local DRAM and a parity error occurs. If the MC68030 is the local bus master when the parity error occurs, the Parity Error (PE) status bit is set in the PCC status register. Note that this bit is only useful if mode 3 parity checking is set. If mode 2 parity checking is set, the MC68030 is not able to read status after the occurrence of the parity error.

Bus Error Processing

Because different conditions can cause bus error exceptions, the software must be able to distinguish the source. To aid in this, the MVME147 provides status bits in the VMEchip and PCC chip.

Generally, the bus error handler can interrogate the status bits and proceed with the result. However, two conditions can corrupt the status bits:

- An interrupt can happen during the execution of the bus error handler (before an instruction can write to the status register to raise the interrupt mask). If the interrupt service routine causes a second bus error, the status that indicates the source of the first bus error may be lost. The software must be written to deal with this. The PCC can be programmed to generate a Level 7 interrupt when a bus error occurs. This may help force the MC68030 to a known place when a bus error occurs.
- The PCC can take a VMEbus bound BERR* (which updates the status bits) between the MC68030 receiving and handling of a bus error, or vice-versa.

MVME147 Support of MC68030 Indivisible Cycles

The MC68030 performs operations that require indivisible cycle sequences to the local DRAM and to the VMEbus. The MVME147 requires special circuitry to support these operations. Indivisible accesses to a single address are called Single Address Read-Modify-Write Cycles (SARMC). Indivisible accesses to multiple addresses are called Multiple Address Read-Modify-Write Cycles (MARMC).

SARMC cycles (caused by Test and Set (TAS) and single byte Compare and Swap (CAS) instructions) are supported fully by the MVME147. This is possible because the VMEbus defines such cycles.

MARMC cycles (caused by CAS2 and multi-byte CAS instructions and by MMU table walking) are conditionally supported by the MVME147. The VMEbus does not define these cycles.

The WAITRMC bit in the PCC controls the support of MARMC cycles. If WAITRMC is cleared, MARMC cycles are not guaranteed to be indivisible. Furthermore, if MARMC cycles straddle onboard DRAM and VMEbus memory, the MVME147 malfunctions.

If WAITRMC is set, MARMC cycles are guaranteed to be indivisible only if the other VMEbus board implements its MARMC cycles the same way as the MVME147 (with WAITRMC set). Note that setting the WAITRMC bit can be a performance penalty. When the bit is set, the MVME147 waits to become VMEbus master before it executes any MARMC cycle (even though it may be going only to onboard DRAM).

EIA-232-D Interconnections A

Introduction

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

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

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

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

Table A-1. EIA-232-D Interconnections

Pin Number	Signal Mnemonic	Signal Name and Description
1		CHASSIS GROUND. Not always used. See section *Proper Grounding.*
2	TxD	TRANSMIT DATA. Data to be transmitted; input to the modem from the terminal.
3	RxD	RECEIVE DATA. Data which is demodulated from the receive line; output from the modem to the terminal.
4	RTS	REQUEST TO SEND. Input to the modem from the terminal when required to transmit a message. With RTS off, the modem carrier remains off. When RTS is turned on, the modem immediately turns on the carrier.
5	CTS	CLEAR TO SEND. Output from the modem to the terminal to indicate that message transmission can begin. When a modem is used, CTS follows the off-to-on transition of RTS after a time delay.
6	DSR	DATA SET READY. Output from the modem to the terminal to indicate that the modem is ready to transmit data.
7	SIG-GND	SIGNAL GROUND. Common return line for all signals at the modem interface.
8	DCD	DATA CARRIER DETECT. Output from the modem to the terminal to indicate that a valid carrier is being received.
9-14		Not used.
15	TxC	TRANSMIT CLOCK (DCE). Output from the modem to the terminal; clocks data from the terminal to the modem.
16		Not used.
17	RxC	RECEIVE CLOCK. Output from the modem to the terminal; clocks data from the modem to the terminal.
18, 19		Not used.
20	DTR	DATA TERMINAL READY. Input to the modem from the terminal; indicates that the terminal is ready to send or receive data.
21		Not used.

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

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

Levels of Implementation

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

- Automatic dialing
- Automatic answering
- Synchronous transmission

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

Signal Adaptations

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

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

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

Sample Configurations

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

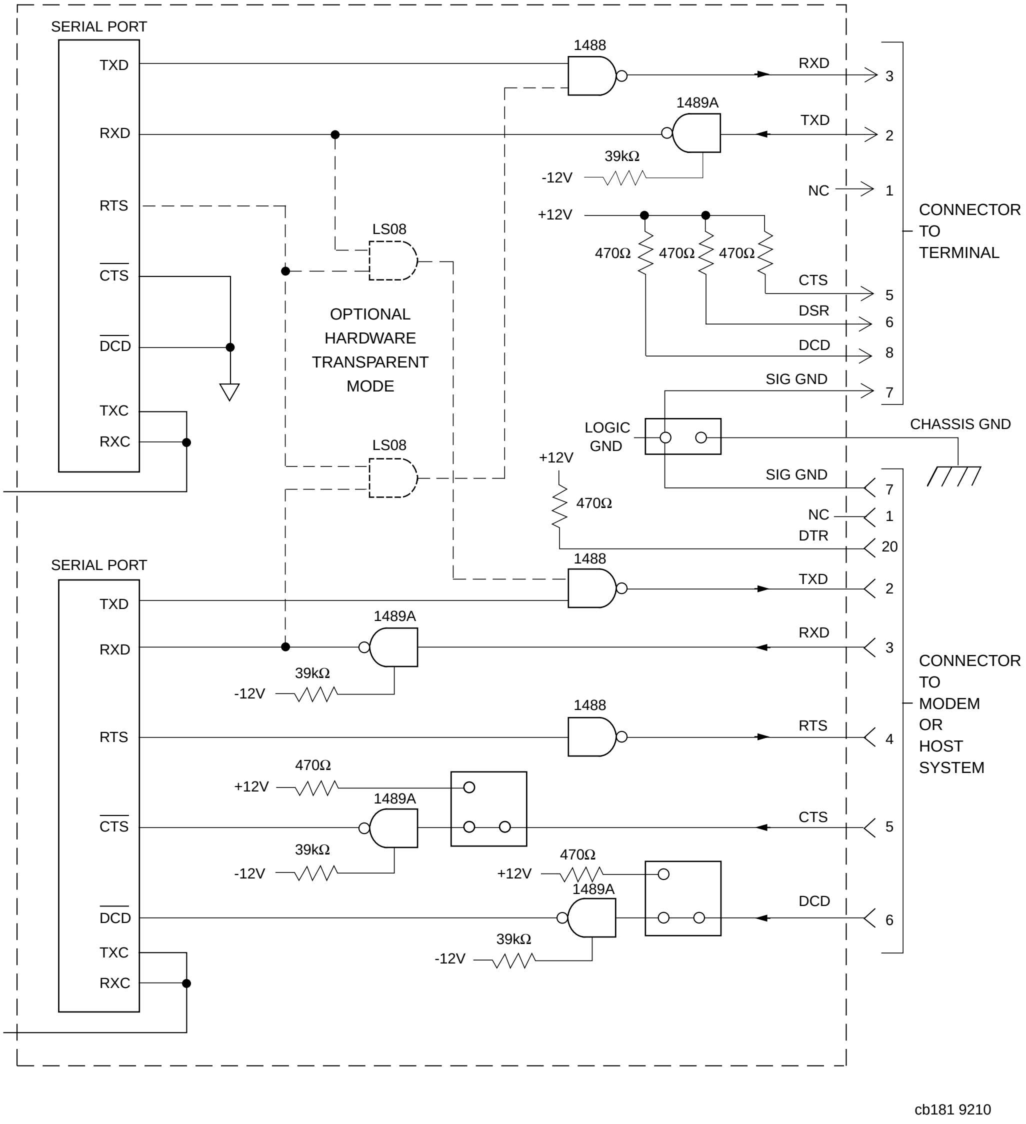


Figure A-1. Middle-of-the-Road EIA-232-D Configuration

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

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

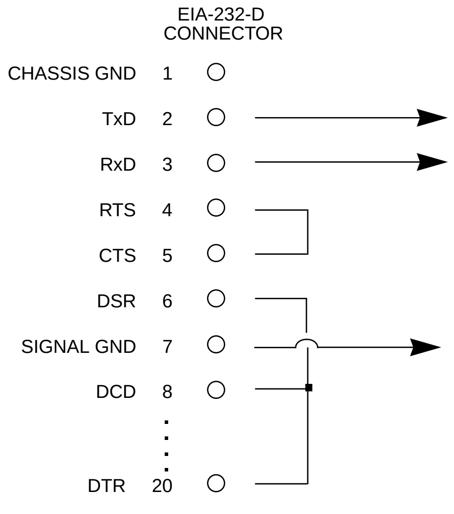


Figure A-2. Minimum EIA-232-D Connection

Proper Grounding

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

The problem is that when units are connected to different electrical outlets, there may be several volts of difference in ground potential. If pin 1 of each device is interconnected with the others via cable, several amperes of current could result. This condition may not only be dangerous for the small wires in a typical cable, but may also produce electrical noise that causes errors in data transmission. That is why Figure A-1 shows no connection for pin 1.

Normally, pin 7 should only be connected to the CHASSIS GROUND at one point; if several terminals are used with one computer, the logical place for that point is at the computer. The terminals should not have a connection between the logic ground return and the chassis.

Debugger General Information B

Overview of M68000 Firmware

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

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

This member of the M68000 firmware family is implemented on the MVME147 MPU VMEmodule and is known as MVME147Bug, or simply 147Bug.

Description of 147Bug

The MVME147Bug package is a powerful evaluation and debugging tool for systems built around the MVME147 monoboard microcomputer. Facilities are available for loading and executing user programs under complete operator control for system evaluation. 147Bug includes:

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

- ❑ Various 147Bug routines that handle I/O, data conversion, and string functions available to user programs through the TRAP #15 system calls

Caution

When using the 147Bug TRAP #15 functions, the interrupt mask is raised to level 7 and the MMU is disabled during the TRAP #15 function.

In addition, 147Bug provides as an option a “system” mode that allows autoboot on power-up or reset, and a menu interface to several system commands used in VME Delta Series systems.

147Bug consists of three parts:

- ❑ A command-driven user-interactive software debugger, described in this appendix, and hereafter referred to as “the debugger” or “147Bug”
- ❑ A command-driven diagnostic package for the MVME147 hardware, hereafter referred to as “the diagnostics”
- ❑ A user interface which accepts commands from the system console terminal

When using 147Bug, you operate out of the debugger directory or the diagnostic directory.

If you are in the debugger directory, the debugger prompt “`147-Bug>`” displays and you have all of the debugger commands at your disposal.

If you are in the diagnostic directory, the diagnostic prompt “`147-Diag>`” displays and you have all of the diagnostic commands at your disposal as well as all of the debugger commands.

You may switch between directories by using the Switch Directories (**SD**) command. You may examine the commands in the particular directory that you are currently in by using the Help (**HE**) command.

Because 147Bug is command-driven, it performs its various operations in response to user commands entered at the keyboard. When you enter a command, 147Bug executes the command and the prompt reappears. However, if you enter a command that causes execution of user target code (e.g., **GO**), then control may or may not return to 147Bug, depending on the outcome of the user program.

The commands are more flexible and powerful than previous debuggers. Also, the debugger in general is more “user-friendly”, with more detailed error messages (refer to *MVME147BUG 147Bug Debugging Package User’s Manual*) and an expanded online help facility.

If you have used one or more of Motorola's other debugging packages, you will find the 147Bug very similar. Considerable effort has also been made to make the interactive commands more consistent. For example, delimiters between commands and arguments may now be commas or spaces interchangeably.

147Bug Implementation

147Bug is physically contained in two 32-pin DIP EPROMs, providing 256KB of storage. Both EPROMs are necessary regardless of how much space is actually occupied by the firmware, because one device is for the even bytes and one is for the odd bytes.

147Bug is written in assembler code, providing the benefits of maintainability and position-independence.

The executable code is checksummed at every power-on, and the result (which includes a pre-calculated checksum contained in the EPROMs) is tested for an expected zero. Thus, if the contents of the EPROMs are modified, the checksum must also be modified.

Caution

When modifying the EPROMs, you must take re-checksum precautions.

The power-on defaults for the MVME147 debug port are:

- ❑ Eight bits per character
- ❑ One stop bit per character
- ❑ Parity disabled (no parity)
- ❑ Baud rate 9600 baud (default baud rate of all MVME147 ports at power-up)

After power-up, the baud rate of the debug port can be reconfigured by using the Port Format (**PF**) command of the 147Bug debugger.

Autoboot

Autoboot is a software routine that can be enabled by a flag in the battery backed-up RAM to provide an independent mechanism for booting an operating system. When enabled by the Autoboot (**AB**) command, this Autoboot routine automatically starts a boot from the controller and device specified. It also passes on the specified default string. This normally occurs at power-up only, but you may change it to boot up at any board reset. **NOAB** disables the routine but does not change the specified parameters.

The Autoboot enable/disable command details are described in *MVME147BUG 147Bug Debugging Package User's Manual*. The default (factory-delivered) condition is with autoboot disabled.

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

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

Following this message there is a delay while the debug firmware waits for the various controllers and drives to come up to speed. 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, hit the BREAK key.

ROMboot

This function is enabled by the ROMboot (**RB**) command and executed at power-up (optionally also at reset), assuming there is valid code in the ROMs (or optionally elsewhere on the module or VMEbus) to support it. If ROMboot code is installed and the environment has been set for Bug mode (refer to Appendix C, *SET and ENV Commands*), a user-written routine is given control (if the routine meets the format requirements).

The **NORB** command disables the ROMboot function.

One use of ROMboot might be resetting SYSFAIL* on an unintelligent controller module.

For your module to gain control through the ROMboot linkage, four requirements must be met:

1. Power must have just been applied (but the **RB** command can change this to also respond to any reset).
2. Your routine must be located within the MVME147 ROM memory map (but the **RB** command can change this to any other portion of the onboard memory, or even offboard VMEbus memory).
3. The ASCII string “BOOT” must be located within the specified memory range.
4. Your routine must pass a checksum test, which ensures that this routine was really intended to receive control at power-up.

To prepare a module for ROMboot, use the Checksum (**CS**) command. When the module is ready it can be loaded into RAM, and the checksum generated, installed, and verified with the **CS** command.

The format of the beginning of the routine is as follows:

Module Offset	Length	Contents	Description
$00	4 bytes	BOOT	ASCII string indicating possible routine.
$04	4 bytes	Entry Offset	Longword offset from "BOOT".
$08	4 bytes	Routine Length	Longword, includes length from module offset $00 to and including checksum.
$0C	?	Routine Name	ASCII string containing routine name.

By convention within Motorola, the checksum is placed in the two bytes following the routine.

If you wish to make use of ROMboot you do not have to fill a complete ROM. Any partial amount is acceptable, as long as the length reflects where the checksum is correct.

ROMboot searches for possible routines starting at the start of the memory map first and checks for the "BOOT" indicator. Two events are of interest for any location being tested:

1. The map is searched for the ASCII string "BOOT".
2. If the ASCII string "BOOT" is found, it is still undetermined whether the routine is meant to gain control. To verify that this is the case, the bytes starting from the beginning of "BOOT" through the end of the routine (as defined by the 4-byte length at offset $8) are run through the checksum routine. If both the even and odd bytes are zero, it is established that the routine was meant to be used for ROMboot.

Under control of the **RB** command, the sequence of searches for "BOOT" is as follows:

1. Search direct address (as set by the **RB** command).
2. Search your non-volatile RAM (first 1KB of battery back-up RAM).
3. Search complete ROM map.

4. Search local RAM (if **RB** command has selected to operate on any reset), at all 8K byte boundaries starting at $00006000.
5. Search the VMEbus map (if so selected by the **RB** command) on all 8K byte boundaries starting at the end of the onboard RAM.

The following example performs these tasks:

1. Outputs a (**CR**)(**LF**) sequence to the default output port.
2. Displays the date and time from the current cursor position.
3. Outputs two more (**CR**)(**LF**) sequences to the default output port.
4. Returns control to 147Bug.

Example:

The target code is first assembled and linked, leaving $00 in the even and odd locations destined to contain the checksum.

Load the routine into RAM (with S-records via the **LO** command, from a disk using **IOP**, or by hand using the **MM** command):

```
147-Bug>mds 6000                           Display entire module (zero checksums
                                           at $0000602C and $0000602D

00006000 424F 4F54 0000 0018  0000 002E 5465 7374   BOOT........Test
00006010 2052 4F4D 424F 4F54  4E4F 0026 4E4F 0052    ROMBOOTNO.&NO.R
00006020 4E4F 0026 4E4F 0026  4E4F 0063 0000 0000   NO.&NO.&NO.c....
00006030 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006040 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006050 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006060 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006070 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006080 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006090 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060A0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060B0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060C0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060D0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060E0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060F0 0000 0000 0000 0000  0000 0000 0000 0000   ................
```

```
147-Bug>md 6018;di                                Disassemble executable instructions
00006018 4E4F0026          SYSCALL          .PCRLF
0000601C 4E4F0052          SYSCALL          .RTC_DSP
00006020 4E4F0026          SYSCALL          .PCRLF
00006024 4E4F0026          SYSCALL          .PCRLF
00006028 4E4F0063          SYSCALL          .RETURN
0000602C 00000000          ORI.B            #$0,D0
00006030 00000000          ORI.B            #$0,D0
00006034 00000000          ORI.B            #$0,D0

147-Bug>CS 6000 602E                              Perform checksum on locations 6000
Effective address: 00006000                       through 602E (refer to CS command)
Effective address: 0000602D
(Even/Odd) = F99F

147-Bug> M 602C;B                                 Insert checksum into bytes $602C,$602D
0000602C 00? F9
0000602D 00? 9F.
147-Bug>CS 6000 602E
Effective address: 00006000                       Verify that the checksum is correct
Effective address: 0000602D
(Even/Odd) = 0000

147-Bug>mds 6000                       Again display entire module (now with checksums)
00006000 424F 4F54 0000 0018  0000 002E 5465 7374   BOOT........Test
00006010 2052 4F4D 424F 4F54  4E4F 0026 4E4F 0052    ROMBOOTNO.&NO.R
00006020 4E4F 0026 4E4F 0026  4E4F 0063 F99F 0000   NO.&NO.&NO.cy...
00006030 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006040 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006050 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006060 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006070 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006080 0000 0000 0000 0000  0000 0000 0000 0000   ................
00006090 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060A0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060B0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060C0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060D0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060E0 0000 0000 0000 0000  0000 0000 0000 0000   ................
000060F0 0000 0000 0000 0000  0000 0000 0000 0000   ................
147-Bug>
```

The routine is now recognized by the ROMboot function when it is enabled by the **RB** command.

Restarting the System

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

Reset

Pressing and releasing the MVME147 front panel **RESET** switch initiates a Reset. COLD and WARM reset modes are available. By default, 147Bug is in COLD reset mode (refer to the **RESET** command description).

During COLD reset mode, a total board initialization takes place, as if the MVME147 had just been powered up. The breakpoint table and offset registers are cleared. The user registers are invalidated. Input and output character queues are cleared. Onboard devices (timer, serial ports, etc.) are reset. All static variables (including disk device and controller parameters) are restored to their default states. Serial ports are reconfigured to their default state.

During WARM reset mode, the 147Bug variables and tables are preserved, as well as the user registers and breakpoints.

If the particular MVME147 is the system controller, then a System Reset is issued to the VMEbus and other modules in the system are reset as well.

The Local Reset feature (the MVME147 is NOT the system controller) is a partial System Reset, not a complete System Reset such as power-up or SYSRESET. When the Local Bus Reset signal is asserted, a local bus cycle may be aborted. Because the VMEchip is connected to both the local bus and the VMEbus, if the aborted cycle is bound for the VMEbus, erratic operation may result. Communications between the local processor and the VMEbus should be terminated by an Abort; Reset should be used only when the local processor is halted or the local bus is hung and Reset is the last resort.

Reset must be used if the processor ever halts (as evidenced by the MVME147 illuminated **STAT** LED), for example after a double bus fault; or if the 147Bug environment is ever lost (vector table is destroyed, etc.).

Abort

Abort is invoked by pressing and releasing the **ABORT** switch on the MVME147 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. (When working in the debugger, Abort captures and stores only the program counter, status register, and format/vector information.) 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, stack pointers, etc., help to pinpoint the malfunction.

Abort generates a level seven interrupt (non-maskable). The target registers, reflecting the machine state at the time the **ABORT** switch was pushed, are displayed to the screen. Any breakpoints installed in your code are removed and the breakpoint table remains intact. Control is returned to the debugger.

Reset and Abort - Restore Battery Backed Up RAM

Pressing both the **RESET** and **ABORT** switches at the same time and releasing the **RESET** switch before the **ABORT** switch initiates an onboard reset and a restore of key Bug-dependent BBRAM variables.

During the start of the Reset sequence, if abort is invoked, then the following conditions are set in BBRAM:

- ❑ SCSI ID is set to 7.
- ❑ Memory sized flag is cleared (onboard memory is sized on this reset).
- ❑ AUTOboot is turned off.
- ❑ ROMboot is turned off.

- ❏ Environment is set for Bug mode.
- ❏ Automatic SCSI bus reset is turned off.
- ❏ Onboard diagnostic switch is turned on (for this reset only).
- ❏ System memory sizing is turned on (System mode).
- ❏ Console is set to port 1 (LUN 0).
- ❏ Port 1 (LUN 0) is set to use ROM defaults for initialization.
- ❏ Concurrent mode is turned off.

In this situation, if a failure occurs during the onboard diagnostics, the **FAIL** LED repeatedly flashes a code to indicate the failure. The on/off LED time for code flashing is approximately 0.25 seconds. The delay between codes is approximately two seconds. To complete bug initialization, press the **ABORT** switch while the LED is flashing. When initialization is complete, a failure message is displayed. LED flashes indicate confidence test failures per the following table.

Number of LED Flashes	Description
1	CPU register test failure
2	CPU instruction test failure
3	ROM test failure
4	Onboard RAM test (first 16KB) failure
5	CPU addressing mode test failure
6	CPU exception processing test failure
7	+12 Vdc fuse failure
10	NVRAM battery low
11	Trouble with the NVRAM
12	Trouble with the RTC

Break

A Break is generated by pressing and releasing the **BREAK** key on the terminal keyboard. Break does not generate an interrupt. The only time Break is recognized is when characters are sent or received by the console port. Break removes any breakpoints in your code and keeps the breakpoint table intact. Break does not, however, take a snapshot of the machine state nor does it display the target registers.

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 without overwriting the contents of the target registers, as would be done if Abort were used.

Memory Requirements

The program portion of 147Bug is approximately 256KB of code and consists of the debugger and diagnostic packages. It is contained entirely in EPROM. The EPROM sockets on the MVME147 are mapped starting at location $FF800000. However, the 147Bug code is position-independent and executes anywhere in memory. SCSI firmware code is not position-independent.

147Bug requires a minimum of 16KB of contiguous read/write memory to operate. This memory is the MVME147 on-board RAM and is used for 147Bug stack and static variable space. The rest of on-board RAM is reserved as user space.

Whenever the MVME147 is reset, the following takes place:

- User PC is initialized to the address corresponding to the beginning of the user space ($4000).
- User stack pointers are initialized to addresses within the user space.

Terminal Input/Output Control

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

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

^X	(cancel line)	The cursor is backspaced to the beginning of the line. If the terminal port is configured with the hardcopy or TTY option (refer to the **PF** command), then a carriage return and line feed is issued along with another prompt.
^H	(backspace)	The cursor is moved back one position. The character at the new cursor position is erased. If the hardcopy option is selected, a "/" character is typed along with the deleted character.
^D	(redisplay)	The entire command line as entered so far is redisplayed on the following line.
Delete key	(delete)	Performs the same function as **^H**.

When observing output from any 147Bug 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 147Bug, but you may change them with the **PF** command. In the initialized (default) mode, operation is as follows:

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

Disk I/O Support

147Bug can initiate disk input/output by communicating with intelligent disk controller modules over the VMEbus. Disk support facilities built into 147Bug consist of:

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

Parameters such as these:

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

are kept in tables by 147Bug. Default values for these parameters are assigned at power-up and cold-start reset, but may be altered as described in the section on default parameters, later in this chapter.

Blocks Versus Sectors

The logical block defines the unit of information for disk devices. A disk is viewed by 147Bug as a storage area divided into logical blocks. By default, the logical block size is set to 256 bytes for every block device in the system. You can change the block size on a per device basis with the IOT command.

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

When a disk transfer is requested:

- The start and size of the transfer is specified in blocks.
- 147Bug translates this into an equivalent sector specification and passes the sector specification on to the controller to initiate the transfer.

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

Disk I/O via 147Bug Commands

The 147Bug commands listed in the following paragraphs are provided for disk I/O. Detailed instructions for their use are found in *MVME147BUG 147Bug Debugging Package User's Manual*. When a command is issued to a particular controller LUN and device LUN, these LUNs are remembered by 147Bug so that the next disk command defaults to use the same controller and device.

IOP (Physical I/O to Disk)

IOP allows you to:

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

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

IOT (I/O Teach)

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

IOC (I/O Control)

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

BO (Bootstrap Operating System)

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

BH (Bootstrap and Halt)

BH reads an operating system or control program from a specified device into memory, then returns control to 147Bug. It is used as a debugging tool.

Disk I/O via 147Bug System Calls

All operations that actually access the disk are done directly or indirectly by 147Bug TRAP #15 system calls. (The command-level disk operations provide a convenient way of using these system calls without writing and executing a program).

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

.DSKRD	Disk read. Use this system call to read blocks from a disk into memory.
.DSKWR	Disk write. Use this system call to write blocks from memory onto a disk.
.DSKCFIG	Disk configure. Use this system call to change the configuration of the specified device.
.DSKFMT	Disk format. Use this system call to send a format command to the specified device.
.DSKCTRL	Disk control. Use this system call to implement any special device control functions that cannot be accommodated easily with any of the other disk functions.

Refer to the *MVME147BUG 147Bug Debugging Package User's Manual* for information on using these and other system calls.

To perform a disk operation, 147Bug must eventually present a particular disk controller module with a controller command packet which has been especially prepared for that type of controller module. (This is accomplished in the respective controller driver module.) A command packet for one type of controller module usually does not have the same format as a command packet for a different type of module. The system call facilities which perform disk I/O:

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

Refer to the system call descriptions in the *MVME147BUG 147Bug Debugging Package User's Manual* for details on the format and construction of these standardized "user" packets.

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

Default 147Bug Controller and Device Parameters

The **IOT** command, with the **T** (teach) option specified, must be invoked to initialize the parameter tables for available controllers and devices. This option instructs IOT to scan the system for all currently supported disk/tape controllers and build a map of the available controllers. This map is built in the Bug RAM area, but can also be saved in NVRAM if so instructed. If the map is saved in NVRAM, then after a reset, the map residing in NVRAM is copied to the Bug RAM area and used as the working map. If the map is not saved in NVRAM, then the map is temporary and the **IOT;T** command must be invoked again if a reset occurs.

If the device is formatted and has a configuration area, then during the first device access or during a boot, **IOT** is not required. Reconfiguration is done automatically by reading the configuration area from the device, then the discriptor for the device is modified according to the parameter information contained in the configuration area.

If the device is not formatted or of unknown format, or has no configuration area, then before attempting to access the device, you should verify the parameters, using **IOT**. The **IOT** command may be used to manually reconfigure the parameter table for any controller and/or device that is different from the default. These are temporary changes and are overwritten with default parameters, if a reset occurs.

The **IOT;T** command should also be invoked any time the controllers are changed or when ever the NVRAM map has been damaged or not initialized ("`No Disk Controllers Available`" is displayed when the **IOT;H** command is invoked).

Disk I/O Error Codes

147Bug returns an error code if an attempted disk operation is unsuccessful.

Multiprocessor Support

The MVME147 dual-port RAM feature makes the shared RAM available to remote processors as well as to the local processor.

A remote processor can initiate program execution in the local MVME147 dual-port RAM by issuing a remote **GO** command using the Multiprocessor Control Register (MPCR). The MPCR, located at shared RAM location base address plus $800, contains one of two longwords used to control communication between processors. The MPCR contents are organized as follows:

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

The codes stored in the MPCR are of two types:

- ❑ Status returned (from 147Bug)
- ❑ Command set by the bus master (job requested by some processor)

The status codes that may be returned from 147Bug are:

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

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

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

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

Base Address + $804	MSB	*	*	LSB	MPAR

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

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

If no terminal is connected to the port, the MPCR is still polled to see whether an external processor requires control to be passed to the dual-port RAM. If a terminal does respond, the MPCR is polled for the same purpose while the serial port is being polled for your 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 previously set breakpoints are enabled when 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 the execution address. (Any remote processor could examine the MPCR contents.)

If the code being executed 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.

Diagnostic Facilities

Included in the 147Bug package is a complete set of hardware diagnostics intended for testing and troubleshooting of the MVME147. These diagnostics are listed in the following tables. In order to use the diagnostics, you must switch directories to the diagnostic directory. If you are in the debugger directory, you can switch to the diagnostic directory by entering the debugger command Switch Directories (**SD**). The diagnostic prompt appears:

```
147-Diag>
```

Table 5-1. Diagnostic Monitor Commands/Prefixes

Command/ Prefix	Description
HE	Help menu command
ST, SST	Self test prefix/command
SD	Switch directories command
LE	Loop-on-error mode prefix
SE	Stop-on-error mode prefix
LC	Loop-continue mode prefix
NV	Non-verbose mode prefix
DE	Display error counters command
ZE	Clear (zero) error counters command
DP	Display pass count command
ZP	Zero pass count command

Table 5-2. Diagnostic Utilities

Command	Description
WL.*size*	Write loop enable
RL.*size*	Read loop enable
WR.*size*	Write/read loop enable

Note: *size* may be **B** (byte), **W** (word), or **L** (longword).

Table 5-3. Diagnostic Test Commands

Command	Description
MPU	MPU tests for the MC68030
CA30	MC68030 onchip cache tests
MT	Memory tests
MMU	Memory Management Unit tests
RTC	Real-time clock tests
BERR	Bus error test
FPC	Floating-point coprocessor (MC68882) test
LAN	LANCE chip (AM7990) functionality test
LANX	LANCE chip (AM7990) external test
SCC	Z8530 functionality test
PCC	Peripheral channel controller functionality test
VMEGA	VME gate array test

Refer to the *MVME147BUG 147Bug Debugging Package User's Manual* for complete descriptions of the diagnostic routines available and instructions on how to invoke them. Note that some diagnostics depend on restart defaults that are set up only in a particular restart mode. Refer to the documentation on a particular diagnostic for the correct mode.

Using the 147Bug Debugger

Entering Debugger Command Lines

147Bug is command-driven and performs its various operations in response to user commands entered at the keyboard. When the debugger prompt (`147-Bug>`) appears on the terminal screen, then the debugger is ready to accept commands.

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

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

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

a. The command identifier (for example, **MD** or **md** for the Memory Display command). Either uppercase or lowercase is allowed.

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

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

d. Any required arguments, as specified by command.

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

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

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

Syntactic Variables

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

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

Expression as a Parameter

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

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

Numeric values may be expressed in either:

- ❑ Hexadecimal
- ❑ Decimal
- ❑ Octal
- ❑ 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 performed according to the following rules:

- ❑ Always evaluated 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 Hexadecimal)**	**Notes**
FF0011	FF0011	
45+99	DE	
&45+&99	90	
@35+@67+@10	5C	
%10011110+%1001	A7	
88<<4	880	shift left
AA&F0	A0	logical AND

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

Address as a Parameter

Many commands use *addr* as a parameter. The syntax accepted by 147Bug is similar to the one accepted by the MC68030 one-line assembler. All control addressing modes are allowed. An “address + offset register” mode is also provided.

Address Formats

Table B-4 summarizes the address formats which are acceptable for address parameters in debugger command lines.

Table B-4. Debugger Address Parameter Formats

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

Table B-4. Debugger Address Parameter Formats (continued)

Notes

1.

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*

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

Offset Registers

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

- ❑ Base
- ❑ Top

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

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

Example:

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

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

******  TOTAL ERRORS     0––
******  TOTAL WARNINGS   0––
```

The above program was loaded at address $0001327C.

The disassembled code is shown next:

```
147Bug>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
147Bug>
```

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

```
147Bug>OF R0
R0 =00000000 00000000? 1327C.
147Bug>MD 0+R0;DI
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
147Bug>
```

For additional information about the offset registers, refer to the *MVME147BUG 147Bug Debugging Package User's Manual*.

Port Numbers

Some 147Bug commands give you the option of choosing the port which is to be used to input or output. The valid port numbers which may be used for these commands are:

0 - MVME147 EIA-232-D (MVME712/MVME712M serial port 1)
1 - MVME147 EIA-232-D (MVME712/MVME712M serial port 2)
2 - MVME147 EIA-232-D (MVME712/MVME712M serial port 3)
3 - MVME147 EIA-232-D (MVME712/MVME712M serial port 4)
4 - MVME147 Printer Port (MVME712/MVME712M printer)

Note These logical port numbers (0, 1, 2, 3, and 4) are referred to as "Serial Port 1", "Serial Port 2", "Serial Port 3", "Serial Port 4", and "Printer Port", respectively, by the MVME147 hardware documentation and by the MVME712/MVME712M hardware documentation.

For example, the command DU1 (Dump S-records to Port 1) would actually output data to the device connected to the serial port labeled SERIAL PORT 2 on the MVME712/MVME712M panel.

Entering and Debugging Programs

There are various ways to enter a user program into system memory for execution. One way is to create the program using the Memory Modify (**MM**) command with the assembler/disassembler option. You enter the program one source line at a time. After each source line is entered, it is assembled and the object code loads into memory. Refer to the *MVME147 BUG 147Bug Debugging Package User's Manual* for complete details of the 147Bug Assembler/ Disassembler.

Another way to enter a program is to download an object file from a host system. The program must be in S-record format (described in the *MVME147BUG 147Bug Debugging Package User's Manual*) and may have been assembled or compiled on the host system. Alternately, the program may have been previously created using the 147Bug **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 MVME147. The file is downloaded from the host to MVME147 memory by the Load (**LO**) command.

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

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

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

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

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 147Bug routines via one of the MC68030 TRAP instructions, using vector #15. Refer to the *MVME147BUG 147Bug Debugging Package 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. 147Bug uses certain of the MVME147 onboard resources and uses onboard memory to contain temporary variables, exception vectors, etc. If you disturb resources upon which 147Bug depends, then the debugger may function unreliably or not at all.

147Bug Vector Table and Workspace

As described in the *Memory Requirements* section in this appendix, 147Bug needs 16KB of read/write memory to operate. The 147Bug reserves a 1024-byte area for a user program vector table area and then allocates another 1024-byte area and builds an exception vector table for the debugger itself to use. Next, 147Bug:

- Reserves space for static variables
- Initializes these static variables to predefined default values
- Allocates space for the system stack
- Initializes the system stack pointer to the top of this area

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

Tick Timers

The MVME147 uses the PCC tick timer 1 to generate accurate delays for program timing (refer to Chapter 4, *Programming*).

Serial Ports

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

Exception Vectors Used by 147Bug

The exception vectors used by the debugger are listed in Table B-5. These vectors must reside at the specified offsets in the target program's vector table for the associated debugger facilities (breakpoints, trace mode, etc.) to operate.

When the debugger handles one of the exceptions listed in Table B-5, the target stack pointer is left pointing past the bottom of the exception stack frame created; that is, it reflects the system stack pointer values just before the exception occurred. In this way, the operation of the debugger facility (through an exception) is transparent to users.

Table B-5. Exception Vectors Used by 147Bug

Vector Offset	Exception	147Bug Facility
$8	Bus Error	
$10	Illegal instruction	Breakpoints (used by **GO**, **GN**, **GT**)
$24	Trace	Trace operations (such as **T**, **TC**, **TT**)
$108	Level 7 Interrupt	ABORT pushbutton
$BC	TRAP #15	System calls

Example:

Trace one instruction using debugger.

```
147Bug>RD
PC   =00004000 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
D0   =00000000 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00006000
00004000 7055               MOVEQ.L #$55,D0
147Bug>T
PC   =00004002 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
D0   =00000055 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   =00006000
00004002 4E71               NOP
147Bug>
```

Notice that the value of the target stack pointer register (A7) has not changed even though a trace exception has taken place. Your program may either use the exception vector table provided by 147Bug or it may create a separate exception vector table of its own. The two following sections detail these two methods.

Using 147Bug Target Vector Table

The 147Bug 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**, **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 loads into the target-state VBR at power-up or cold-start reset and can be observed by using the **RD** command to display the target-state registers immediately after power-up.

The Bug initializes the target vector table with the debugger vectors listed in Table B-5 and fills the other vector locations with the address of a generalized exception handler (refer to the *Bug Generalized Exception Handler* section in this chapter). The target program may take over as many vectors as desired by simply writing its own exception vectors into the table. If the vector locations listed in Table B-5 are overwritten, the accompanying debugger functions are lost.

The 147Bug 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 own exception vectors. If this is done, the program must change the value of the VBR to point to the new vector table. In order to use the debugger facilities you can copy the proper vectors from the Bug vector table into the corresponding vector locations in your program vector table.

The vector for the Bug generalized exception handler (described in detail in the *Bug Generalized Exception Handler* section in this appendix) may be copied from offset $3C (Uninitialized Interrupt) in the target vector table to all locations in your program vector table where a separate exception handler is not used. This provides diagnostic support in the event that your program is stopped by an unexpected exception. The generalized exception handler gives a formatted display of the target registers and identifies the type of the exception.

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

```
*
***  BUILDX - Build exception vector table ****
*
BUILDX   MOVEC.L    VBR,A0                 Get copy of VBR
         LEA        $10000,A1              New vectors at $10000
         MOVE.L     $3C(A0),D0             Get generalized exception vector
         MOVE.W     $3FC,D1                Load count (all vectors)
LOOP     MOVE.L     D0,(A1,D1)             Store generalized exception vector
         SUBQ.W     #4,D1
         BNE.B      LOOP                   Initialize entire vector table
         MOVE.L     $8(A0),$8(A1)          Copy bus error vector
         MOVE.L     $10(A0),$10(A1)        Copy breakpoints vector
         MOVE.L     $24(A0),$24(A1)        Copy trace vector
         MOVE.L     $BC(A0),$BC(A1)        Copy system call vector
         MOVE.L     $108(A0),$108(A1)      Copy ABORT vector
         LEA.L      COPROCC(PC),A2         Get your exception vector
         MOVE.L     A2,$2C(A1)             Install as F-Line handler
         MOVEC.L    A1,VBR                 Change VBR to new table
         RTS
         END
```

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

When an exception occurs which you want to pass on to the debugger; e.g., 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 Bug target program vector table (which your program saved), yielding the address of the Bug exception vector. The program then jumps to the address stored at this vector location, which is the address of the Bug exception handler.

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

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

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

Bug Generalized Exception Handler

The 147Bug has a generalized exception handler which it uses to handle all of the exceptions not listed in Table B-5. For all these exceptions, the target stack pointer is left pointing to the top of the exception stack frame created. In this way, if an unexpected exception occurs during execution of your code, you are presented with the exception stack frame to help determine the cause of the exception. The following example illustrates this:

Example:

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

```
147Bug>RD
PC   =00004000 SR   =2700=TR:OFF_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00006000 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
D0   =00000000 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00006000
00004000 203900F0           MOVE.L      ($F00000).L,D0
147Bug>T

VMEbus Error


Exception: Long Bus Error
Format/Vector=B008
SSW=074D Fault Addr.=00F00000 Data In=FFFFFFFF Data Out=00004006
PC   =00004000 SR   =A700=TR:ALL_S._7_.....  VBR  =00000000
USP  =00005830 MSP  =00005C18 ISP* =00005FA4 SFC  =0=F0
CACR =0=D:...._I:...          CAAR =00000000 DFC  =0=F0
D0   =00000000 D1   =00000000 D2   =00000000 D3   =00000000
D4   =00000000 D5   =00000000 D6   =00000000 D7   =00000000
A0   =00000000 A1   =00000000 A2   =00000000 A3   =00000000
A4   =00000000 A5   =00000000 A6   =00000000 A7   =00005FA4
00004000 203900F0           MOVE.L      ($F00000).L,D0
147Bug>
```

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

```
147Bug>MD (A7):&44
00005FA4 A700 0000 4000 B008  3EEE 074D FFFF 094E    '...@.0.>n.M...N
00005FB4 00F0 0000 00F0 0000  0000 35EC 2039 0000    .p...p....5l 9..
00005FC4 0000 400A 0000 4008  0000 4006 FFFF FFFF    ..@...@...@.....
00005FD4 00F0 0000 100F F487  0000 A700 FFFF FFFF    .p....t...'.....
00005FE4 0000 7FFF 0000 0000  9F90 0000 0000 6000    ..............'.
00005FF4 0000 0000 0000 0000                         ........
147Bug>
```

Memory Management Unit Support

The Memory Management Unit (MMU) is supported in 147Bug. An MMU confidence check is run at power-up to verify that the registers can be accessed. It also ensures that a context switch can be done successfully. The commands **RD**, **RM**, **MD**, and **MM** have been extended to allow display and modification of MMU data in registers and in memory. MMU instructions can be assembled/disassembled with the **DI** option of the **MD**/**MM** commands. In addition, the MMU target state is saved and restored along with the processor state as required when switching between the target program and 147Bug. Finally, there is a set of diagnostics to test functionality of the MMU.

At power-up, an MMU confidence check is executed. If an error is detected the test is aborted and the message "`MMU failed test`" is displayed. If the test runs without errors then the message "`MMU passed test`" is displayed and an internal flag is set. This flag is later checked by the bug when doing a task switch. The MMU state is saved and restored only if this flag is set.

The MMU defines the Double Longword (DL) data type, which is used when accessing the root pointers. All other registers are either byte, word, or longword registers.

The MMU registers are shown below, along with their data types in parentheses:

Address Translation Control (ATC) Registers:

CRP	CPU Root Pointer Register	(DL)
SRP	Supervisor Root Pointer Register	(DL)
TC	Translation Control Register	(L)
TT0	Transparent Translation 0	(L)
TT1	Transparent Translation 1	(L)

Status Information Registers:

MMUSR	MMU Status Register	(W)

For more information about the MMU, refer to the *MC68030 Enhanced 32-Bit Microprocessor User's Manual*.

Function Code Support

The function codes identify the address space being accessed on any given bus cycle, and in general, they are an **extension** of the address. This becomes more obvious when using a memory management unit, because two identical logical addresses can be made to map to two different physical addresses. In this case, the function codes provide the additional information required to find the proper memory location.

For this reason, the following debugger commands were changed to allow the specification of function codes:

MD	Memory display
MM	Memory modify
MS	Memory set
GO	Go to target program
GD	Go direct (no breakpoints)
GT	Go and set temporary breakpoint
GN	Go to next instruction
BR	Set breakpoint

The symbol **^** (up arrow or caret) following the address field indicates that a function code specification follows. The function code can be entered by specifying a valid function code mnemonic or by specifying a number between 0 and 7. The syntax for an address and function code specification is:

addr^FC

The valid function code mnemonics are:

Function Code	Mnemonic	Description
0	F0	Unassigned, reserved
1	UD	User Data
2	UP	User Program
3	F3	Unassigned, reserved
4	F4	Unassigned, reserved
5	SD	Supervisor Data
6	SP	Supervisor Program
7	CS	CPU Space Cycle

Notes Using an unassigned or reserved function code or mnemonic results in a Long Bus Error message.

If the symbol **^** (up arrow or caret) is used without a function code or mnemonic, the function code display is turned off.

Example:

Change data at location $5000 in your data space.

```
147Bug>M 5000^ud
00005000^UD 0000 ? 1234.
147Bug>
```

The 147Bug Debugger Command Set

The 147Bug debugger commands are summarized in Table B-6. The command syntax is shown in the table using the symbols explained in the section *Using the 147Bug Debugger*, beginning on page B-23.

HE is the 147Bug help facility:

- ❑ Typing **HE** displays the command names of all available commands along with their appropriate titles.
- ❑ Typing **HE** *command* displays the command name and title for that particular command.

The **SET** and **ENV** commands are described in Appendix C.

All other command details are explained in the *MVME147BUG 147Bug Debugging Package User's Manual*.

Table B-6. Debugger Commands

Command Mnemonic	Title	Command Line Syntax
AB	Automatic Bootstrap Operating System	**AB**
NOAB	No Autoboot	**NOAB**
BC	Block of Memory Compare	**BC** *range del addr* [**;** **B** ǀ **W** ǀ **L**]
BF	Block of Memory Fill	**BF** *range del data* [*increment*] [**;** **B** ǀ **W** ǀ **L**]
BH	Bootstrap Operating System and Halt	**BH** [*del controller lun*] [*del device lun*] [*del string*]
BI	Block of Memory Initialize	**BI** *range* [**;** **B** ǀ **W** ǀ **L**]
BM	Block of Memory Move	**BM** *range del addr* [**;** **B** ǀ **W** ǀ **L**]
BO	Bootstrap Operating System	**BO** [*del controller lun*] [*del device lun*] [*del string*]
BR	Breakpoint Insert	**BR** [*addr* [:*count*]]
NOBR	Breakpoint Delete	**NOBR** [*addr*]
BS	Block of Memory Search	**BS** *range del text* [**;** **B** ǀ **W** ǀ **L**] or **BS** *range del data* [*del mask*] [**;** **B** ǀ **W** ǀ **L** [**,N**][**,V**]]
BV	Block of Memory Verify	**BV** *range del data* [*increment*] [**;** **B** ǀ **W** ǀ **L**]
CS	Checksum	**CS** *range*
DC	Data Conversion	**DC** *exp* ǀ *addr*
DU	Dump S-records	**DU** [*port*] *del range* [*del text*] [*del addr*] [*del offset*] [**;** **B** ǀ **W** ǀ **L**]
EEP	EEPROM Programming	**EEP** *range del addr* [**; W**]
ENV	Set Environment to Bug/ Operating System	**ENV** [**; D**]
G/GO	Go Execute Target Code	**GO** [*addr*]
GD	Go Direct (Ignore Breakpoints)	**GD** [*addr*]

Table B-6. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
GN	Go to Next Instruction	**GN**
GT	Go to Temporary Breakpoint	**GT** *addr [:count]*
HE	Help	**HE** [*command*]
IOC	I/O Control for Disk/Tape	**IOC**
IOP	I/O Physical (Direct Disk Access)	**IOP**
IOT	I/O "TEACH" for Configuring Disk Controller	**IOT** [**;** [**A** | **H** | **T**]]
LO	Load S-records from Host	**LO** [*n*] [*addr*] [**;** **X** | **-C** | **T**] [*=text*]
LSAD	LAN Station Address Display/Set	**LSAD**
MA	Macro Define/Display	**MA** [*name*]
NOMA	Macro Delete	**NOMA** [*name*]
MAE	Macro Edit	**MAE** *name line#* [*string*]
MAL	Enable Macro Expansion Listing	**MAL**
NOMAL	Disable Macro Expansion Listing	**NOMAL**
MAW	Save Macros	**MAW** [*controller lun*] [*del* [*device lun*] [*del block #*]]
MAR	Load Macros	**MAR** [*controller lun*] [*del* [*device lun*] [*del block #*]]
M/MM	Memory Modify	**MM** *addr* [**;** [[**B** | **W** | **L** | **S** | **D** | **X** | **P**][**A**] [**N**]] | [**DI**]]
MD	Memory Display	**MD**[**S**] *addr* [**:** *count* | *addr*] [**;** [**B** | **W** | **L** | **S** | **D** | **X** | **P** | **DI**]]

Table B-6. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
MENU	System Menu	**MENU**
MS	Memory Set	**MS** *addr [hexadecimal #] ...* | *['string']...*
OBA	Set Memory Address from VMEbus	**OBA**
OF	Offset Registers Display/Modify	**OF** [**R***n* [**; A**]]
PA	Printer Attach	**PA** [*n*]
NOPA	Printer Detach	**NOPA** [*n*]
PF	Port Format	**PF** [*port*]
NOPF	Port Detach	**NOPF** [*port*]
PS	Put RTC Into Power Save Mode for Storage	**PS**
RB	ROMboot Enable	**RB**
NORB	ROMboot Disable	**NORB**
RD	Register Display	**RD** {[+|**-**|=] [*dname*] [/]} {[+|**-**|=] [*reg1* [**-***reg2*]] [/]}
REMOTE	Connect the Remote Modem to CSO	**REMOTE**
RESET	Cold/Warm Reset	**RESET**
RM	Register Modify	**RM** [*reg*]
RS	Register Set	**RS** *reg* [*del exp*]
SD	Switch Directories	**SD**
SET	Set Time and Date	**SET**
T	Trace	**T** [*count*]
TA	Terminal Attach	**TA** [*port*]
TC	Trace on Change of Control Flow	**TC** [*count*]

Table B-6. Debugger Commands (Continued)

Command Mnemonic	Title	Command Line Syntax
TIME	Display Time and Date	**TIME**
TM	Transparent Mode	**TM** [*n*] [*escape_key*]
TT	Trace to Temporary Breakpoint	**TT** *addr*
VE	Verify S-records Against Memory	**VE** [*n*] [*addr*] [**; X**|**-C**] [=*text*]

SET and ENV Commands C

Initializing the MVME147

The MVME147 module is shipped with the M48T18 RTC chip's oscillator stopped, to minimize current drain from the onchip battery. A normal cold start of the MVME147, when the 147Bug EPROMs are installed, starts the oscillation; but before you can use your MVME147, you will need to set the time and date correctly with the 147Bug's **SET** command.

The MVME147's NVRAM contains certain operating environment parameters. Before you begin using your MVME147, you should verify these default parameters and/or change them with 147Bug's **ENV** command.

SET - Set Time and Date

Command Input

SET

Description

Begin the **SET** command's interactive dialog by entering **SET** followed by a carriage return **(CR)**. **SET** displays date, time, and calibration values, and prompts for entry of the date in the form MM/DD/YY. To change the displayed date, type a new date followed by (**CR**); to accept the displayed date unchanged, simply enter (**CR**).

Note To correct an incorrect entry, backspace or delete the entire line before pressing the carriage return. When the carriage return is entered, the values are stored in the time-of-day clock.

The next prompt asks for a calibration value. This value slows down (- value) or speeds up (+ value) the RTC in the M48T18 chip. Refer to the M48T18 data sheet (listed in *Related Documentation* in Chapter 1) for details.

The third prompt asks for the time in the form HH:MM:SS. To change the displayed time, type a new time followed by (**CR**); to accept the displayed time unchanged, simply enter (**CR**).

Example: The following example sets a date and time of March 16, 1996, 2:05:32 PM:

```
147-Bug>SET
Weekday  xx/xx/xx    xx:xx:xx
Present calibration = -0
Enter date as MM/DD/YY
03/16/96
Enter Calibration value +/- (0 to 31)
Enter time as HH:MM:SS (24 hour clock)
14:05:32
147-Bug>
```

Related Commands:

To display the current date and time of day, use the **TIME** command. If you need to disable the RTC for storage, enter the **PS** command.

ENV - Set Environment to Bug/Operating System

Command Input

ENV [;D]

Description

The **ENV** command allows you to select the environment in which the debugger is to execute. When you specify **ENV**, the debugger remains in the specified environment until you invoke **ENV** again to change it. The selections are saved in NVRAM and used whenever power is lost.

Note The "Reset and Abort" function sets the environment to the default "Bug" mode until changed by the **ENV** command.

When you invoke the **ENV** command, the interactive mode is entered immediately. The following rules apply while in interactive mode:

	All numerical values are interpreted as hexadecimal numbers.
	When a list is shown, only listed values are accepted. You may use uppercase or lowercase interchangeably.
^	Backs up to the previous option.
.	Entering a period by itself or following a new value/setting causes **ENV** to exit the interactive mode. Control returns to 147Bug.
(CR)	Pressing the carriage return key without entering a value preserves the current value and causes the next prompt to be displayed.

If NVRAM has been corrupted you can repair it by invoking the individual command(s) that correct the bad data, or you can invoke the **ENV** command with the **D** (defaults) option specified. This option instructs **ENV** to update the NVRAM with default environmental parameters.

Using ENV with the Defaults Option

The defaults are defined as follows:

- Bug Mode
- Automatic Bug Self Test is bypassed
- Execute Memory Tests
- Maintain Concurrent Mode through a Power Cycle/Reset
- System Memory Sizing (System Mode only)
- Set the seven VMEchip options to defaults
- No automatic SCSI Bus reset
- SCSI ID set to 7
- Off board Address set to zero
- No ROM-boot and ROM-boot address set to start of ROM
- No Auto-boot
- Set Disk Map to default
- Set console port to zero and all ports use default parameters.

Example 1:

```
147-Bug>env;d
Update with Auto-Configuration Defaults

Update Non-Volatile RAM [Y/N] = N? (CR)
WARNING: Update(s) Discarded
147-Bug>
```

Example 2:

```
147-Bug>ENV;D
Update with Auto-Configuration Defaults

Update Non-Volatile RAM [Y/N] = N? Y

CPU clock frequency [16,20,25,32] = 25? (CR)

Reset System [Y/N] = N?(CR)
WARNING: Updates will not be in effect until a RESET is performed.
147-Bug>
```

Example 3:

```
147-Bug>env;d
Update with Auto-Configuration Defaults

Update Non-Volatile RAM [Y/N] = N? y

CPU clock frequency [16,20,25,32] = 25?(CR)

Reset System [Y/N] = N?y
```

The firmware now takes the reset path and initializes the MVME147 with the defaults placed in NVRAM.

Using ENV without Options

When you invoke **ENV** without the **D** option, you are prompted for the following modes and options:

Two modes are available:

- ❑ **Bug Mod**e. This is the standard mode of operation, and is the one defaulted to if NVRAM should fail.
- ❑ **System Mode**. This is the mode for system operation and is defined in *MVME147BUG 147Bug Debugging Package User's Manual*.

Three Bug Mode options are available:

- ❑ Execute/Bypass Bug Self Test:
  - **Execute**. This mode enables the extended confidence tests as defined in *MVME147BUG 147Bug Debugging Package User's Manual*. This automatically puts the Bug in the diagnostic directory.
  - **Bypass**. In this mode the extended confidence tests are bypassed, this is the mode defaulted to if NVRAM should fail.
- ❑ Execute/Bypass SST Memory Test:
  - **Execute**. This is the standard SST memory test mode, and is the one defaulted to if NVRAM should fail. In this mode the SST memory tests are executed as part of the automatic Bug self test.
  - **Bypass**. In this mode the SST memory tests are bypassed, but the board memory is zeroed at the end of SST to initialize parity.
- ❑ Maintain Concurrent Mode through a Power Cycle/Reset:
  - **Yes**. If Concurrent Mode is entered, a Power Cycle or Reset does not terminate the Concurrent Mode. This is the mode defaulted to if NVRAM should fail.
  - **No**. Power Cycle or Reset causes an exit from Concurrent Mode.

Three System Mode options are available:

- ❑ Execute/Bypass System Memory Sizing:
  - **Execute**. This is the standard mode of operation, and is the one defaulted to if NVRAM should fail. In this mode the System Memory Sizing is invoked during board initialization to find the start and end of contiguous system memory.
  - **Bypass**. In this mode the System Memory Sizing is bypassed and the message `No offboard RAM detected` is displayed.

- ❑ Execute/Bypass SST Memory Test:
  - **Execute.** This is the standard SST memory test mode, and is the one defaulted to if NVRAM should fail. In this mode the SST memory tests are executed as part of the system self test.
  - **Bypass**. In this mode the SST memory tests are bypassed, but the system memory is zeroed at the end of SST to initialize parity.
- ❑ Maintain Concurrent Mode through a Power Cycle/Reset:
  - **Yes**. If Concurrent Mode is entered, a Power Cycle or Reset does not terminate the Concurrent Mode. This is the mode defaulted to if NVRAM should fail.
  - **No**. Power Cycle or Reset causes an exit from Concurrent Mode.

Seven VMEchip options are available, as shown in Table C-1:

Table C-1. VMEchip Options

Option	Description
Board Identification	Allows unique board identification.
GCSR Base Address Offset	Sets the base address of the global control and status register in the VMEbus short I/O map. This value is an offset from the start ($FFFF0000) of the map.
Utility Interrupt Mask	This is used to enable the VMEchip to respond to specific utility interrupt requests. Refer to the *Programming the VMEchip* section for bit definitions and functional descriptions.
Utility Interrupt Vector Number	Interrupt vector number ($8 to $F8) for the utility interrupts. Must be in multiples of $8.
VMEbus Interrupt Mask	This is used to enable the VMEchip to respond to specific VMEbus interrupt requests. Refer to the *Programming the VMEchip* section for bit definitions and functional descriptions.

Table C-1. VMEchip Options (Continued)

Option	Description
VMEbus Requester Level	This is used to configure the VMEbus requester level (0 through 3).
VMEbus Requester Release	This is used to configure the VMEbus requester release mode (Release: On Request, When Done, or Never).

Example 1:

```
147-Bug>env
Bug or System environment [B,S] = B? (CR)                        No change
Execute/Bypass Bug Self Test [E,B] = B? e                        Change to execute
Execute/Bypass SST Memory Test [E,B] = E? (CR)
Maintain Concurrent Mode (if enabled) through a Power Cycle/Reset [Y/N] = Y? (CR)
Set VME Chip:
Board ID (def is 0) [0-FF] = $00? (CR)
GCSR base address offset (def is 0F) [0-0F] = $0F? (CR)
Utility Interrupt Mask (def is 0) [0-FE] = $00? (CR)
Utility Interrupt Vector number (def is 60) [8-F8] = $60? 10 Change vector
VMEbus Interrupt Mask (def is FE) [0-FE] = $FE? (CR)
VMEbus Requester Level (def is 0) [0-3] = 00? (CR)
VMEbus Requester Release (def is ROR) [ROR,RWD,NVR] = ROR? (CR)
147-Bug>
```

Example 2:

```
147-Bug> ENV
Bug or System environment [B,S] = B? (CR)                        No change
Execute/Bypass Bug Self Test [E,B] = E? B                        Change to bypass
Maintain Concurrent Mode (if enabled) through a Power Cycle/Reset [Y/N] = Y? (CR)
Set VME Chip:
Board ID (def is 0) [0-FF] = $00? 2.                             Change and exit
147-Bug>
```

Example 3:

```
147-Bug>ENV
Bug or System environment [B,S] = B? S                           Change to system
Execute/Bypass System Memory Sizing [E,B] = E? (CR)
Execute/Bypass SST Memory Test [E,B] = E? (CR)
Maintain Concurrent Mode (if enabled) through a Power Cycle/Reset [Y/N] = Y? (CR)
Set VME Chip:
```

```
Board ID (def is 0) [0-FF] = $02? 0                                      Change and continue
GCSR base address offset (def is 0F) [0-0F] = $0F? (CR)
Utility Interrupt Mask (def is 0) [0-FE] = $00? (CR)
Utility Interrupt Vector number (def is 60) [8-F8] = $10? (CR)
VMEbus Interrupt Mask (def is FE) [0-FE] = $FE? ^                        Back up
Utility Interrupt Vector number (def is 60) [8-F8] = $10? 60.
                                                                         Change and exit
147-Bug>
```

Firmware now takes the Reset path and initializes the MVME147 for the System Mode (refer to *MVME147BUG 147Bug Debugging Package User's Manual* for System Mode operation details).

Troubleshooting: Solving Start-up Problems D

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

Table D-1. Basic Troubleshooting Steps

Condition ...	Possible problem ...	Try this ...
I. Nothing works, no display on the terminal.	A. If the RUN LED is not lit, the board may not be getting correct power.	1. Make sure the system is plugged in. 2. Check that the board is securely installed in its backplane or chassis. 3. Check that all necessary cables are connected to the board, per this manual. 4. Review the Installation and Start-up procedures in Chapter 2. This includes a step-by-step power-up routine. Try it.
	B. If the LEDs are lit, the board may be in the wrong slot.	1. For VMEmodules, the CPU board should be in the first (leftmost) slot. 2. Also check that the “system controller” function on the board is enabled, per Chapter 2.
	C. The “system console” terminal may be configured wrong.	Configure the system console terminal, per Chapter 2.

Table D-1. Basic Troubleshooting Steps (Continued)

Condition ...	Possible problem ...	Try this ...
II. There is a display on the terminal, but input from the keyboard has no effect.	A. The keyboard may be connected incorrectly.	Recheck the keyboard and power connections.
	B. Board jumpers may be configured incorrectly.	Check the board jumpers per *Hardware Preparation* in Chapter 2.
	C. You may have invoked flow control by pressing a HOLD or PAUSE key, or by typing **<CTRL>-S**	Press the HOLD or PAUSE key again. If this does not free up the keyboard, type in **<CTRL>-Q**
III. Debug prompt `147-Bug>` does not appear at power-up, and the board does not auto boot.	A. Debugger EPROM may be missing.	1. Disconnect *all* power from your system. 2. Check that the proper debugger EPROM is installed per this manual. 3. Reconnect power. **Caution** Performing the next step will change some parameters that may affect your system operation. 4. Restart the system by "double-button reset": press the RESET and ABORT switches at the same time; release RESET first, wait five seconds, then release ABORT. 5. If the debug prompt appears, go to step IV. If the debug prompt does not appear, go to step VI.
	B. The board may need to be reset.	

Table D-1. Basic Troubleshooting Steps (Continued)

Condition ...	Possible problem ...	Try this ...
IV. Debug prompt `147-Bug>` appears at power-up, but the board does not auto boot.	A. The initial debugger environment parameters may be set wrong.	⚠ **Caution** Performing the next step will change some parameters that may affect your system operation.
	B. There may be some fault in the board hardware.	1. Type in **env;d (CR)** This sets up the default parameters for the debugger environment. 2. When prompted to Update Non-Volatile RAM, type in **y (CR)** 3. When prompted for CPU Clock Frequency (in MHz), change it only if it is not correct. 4. When prompted to Reset System, type in **y (CR)** After a cold start, the debug prompt `147-Bug>` is displayed. 5. Change to the diagnostic directory by typing **sd (CR)** Now the prompt should be `147-Diag>`

Table D-1. Basic Troubleshooting Steps (Continued)

Condition ...	Possible problem ...	Try this ...
		6. Run selftest by typing in **st (CR)** The tests take as much as 10 minutes, depending on RAM size. They are complete when the prompt returns. (The onboard selftest is a valuable tool in isolating defects.) 7. The system may indicate that it has passed all the selftests. Or, it may indicate a test that failed. If neither happens, enter **de (CR)** Any errors should now be displayed. If there are any errors, go to step VII. If there are no errors, go to step VI.

V. Debug prompt
`147-Bug>`
appears at power-up, but then the following message appears:
`*** WARNING ***`
`Unreliable R/W to`
`non-volatile RAM`
	The NVRAM device was corrupted and/or replaced, setting bits in a status flag in NVRAM. This flag is normally zero.	! **Caution** Performing the next step will change some parameters that may affect your system operation.
		1. Initialize NVRAM; use the Block Fill command to fill the NVRAM area of the new device with all zeros:
		BF FFFE0000 FFFE07F7 0000
		2. Use the LSAD command to assign the Ethernet address, as described in *Installation Instructions* in Chapter 2.
		Go to Step IV.

Table D-1. Basic Troubleshooting Steps (Continued)

Condition ...	Possible problem ...	Try this ...
VI. The debugger is in System Mode and the board auto boots, or the board has passed selftests.	A. No problems - troubleshooting is done.	No further troubleshooting steps are required. **Note** Even if the board passes all tests, it may still be bad. Selftest does not try out all functions in the board (for example, SCSI, or VMEbus tests).
VII. The board has failed one or more of the tests listed above, and can not be corrected using the steps given.	A. There may be some fault in the board hardware or the on-board debugging and diagnostic firmware.	1. Document the problem and return the board for service. 2. Phone 1-800-222-5640.

Index

INDEX